Herbert Klenk, Hubert B. Keller, Erhard Plödereder,
Peter Dencker
(Hrsg.)

Automotive – Safety & Security 2014 (2015)

Sicherheit und Zuverlässigkeit für automobile Informationstechnik

Tagung
21.-22. 04. 2015 in Stuttgart

Gesellschaft für Informatik e. V. (GI)

Lecture Notes in Informatics (LNI) - Proceedings
Series of the Gesellschaft für Informatik (GI)

Volume P-240

ISBN 978-3-88579-634-3
ISSN 1617-5468

Volume Editors
Herbert Klenk
Airbus Defence and Space GmbH
Rechliner Str.
85077 Manching
Email: herbert.klenk@cassidian.com
Erhard Plödereder
Institut für Softwaretechnologie, Universität Stuttgart
Universitätsstr. 38
70569 Stuttgart
Email: ploedere@informatik.uni-stuttgart.de

printed by Köllen Druck+Verlag GmbH, Bonn

Vorwort

Die 6. Tagung Automotive – Safety & Security 2014 - Sicherheit und Zuverlässigkeit für automobile Informationstechnik fand, nach einer Verlegung vom November 2014 in den April 2015, am 21. - 22. April 2015 in Stuttgart-Feuerbach im Auditorium der Robert Bosch GmbH statt. Die Tagungsserie ist mit der Zuverlässigkeit und Sicherheit softwarebasierter Funktionen im Automotive-Bereich befasst. Zwei Tage lang wurden die neuesten Ideen und konkreten Lösungen für die drängenden Herausforderungen der Softwareentwicklung mit Schwerpunkt auf Sicherheit und Zuverlässigkeit sowie Qualität in den Endprodukten diskutiert. Hochkarätige eingeladene Hauptredner waren Robert C. Seacord und David Oswald. Robert Seacord ist Principal Researcher am Software Engineering Institute der Carnegie Mellon University (CMU) in Pittsburgh, USA, Adjunct Professor an der CMU und bekannt als Verfasser von Sicherheitsrichtlinien des CERT/CC, dessen Secure Coding Initiative er leitet. Er referierte über die zu erwartenden Schwachstellen der Sicherheit von Car-to-X-Kommunikation und wie diese zu vermeiden sind. David Oswald arbeitet als Forscher an der School of Computer Science der University of Birmingham und trug über "Embedded security in practice: About side-channel attacks, reverse-engineering, and more" vor. Direkt nach der Konferenz fanden am 22.4. zwei halbtägige Tutorials von Dr. D. L. Buttle, ETAS GmbH, Stuttgart, zum Thema „Keeping focus: avoiding C's distractions by developing software in ESDL", und von Robert Seacord über „Vulnerabilities in embedded systems" statt. Parallel zur Konferenz zeigte eine Ausstellung einschlägige Werkzeuge rund um die automotive Softwareentwicklung. Der Konferenz bereits am 20. April vorgeschaltet fanden Sitzungen von GI und VDI Fachausschüssen und –gremien, die auch fachliche Träger dieser Konferenz sind, statt. Anschließend traf man sich zu einem Come Together für alle Konferenzteilnehmer.

Softwarebasierte Funktionen stellen den wesentlichen Mehrwertfaktor in nahezu allen Teilen der Industrie, und insbesondere im Automotive-Sektor, dar. Mit vertretbaren Kosten können durch Software erhebliche technische Fortschritte erzielt werden. Wesentlich ist dabei aber, die Zuverlässigkeit gerade in eingebetteten Systemen trotz der wachsenden Komplexität auf anspruchsvollem Niveau zu erhalten. Die Vision von autonomen Fahrzeugen wird durch hochkomplexe Software realisiert werden. Hierbei ist es zwingend notwendig, die Fehler minimal und die Kosten wirtschaftlich vertretbar zu halten. In frühen Phasen eingeführte Softwarefehler haben in der Vergangenheit bei der Wartung zu erheblichen Kosten und zu unvertretbaren Verzögerungen geführt. Hier müssen verbesserte Vorgehensweisen, Methoden und Werkzeuge zur Systematisierung des Entwicklungsprozesses und zur Gewährleistung der nötigen Sicherheit und Zuverlässigkeit eingesetzt werden. Der Personal Software Process aus dem CMMI ist sicher ein notwendiger Schritt. Allerdings muss beachtet werden, dass die hochgradige Vernetzung im und außerhalb des Fahrzeugs zu neuer Komplexität und zu neuen Risiken führt. Kritische Ereignisse in letzter Zeit zeigen drastisch, dass ohne die Berücksichtigung entsprechender Sicherheitsanforderungen sowohl aus dem Safety- als auch aus dem Security-Bereich erhebliche Risiken in der zukünftigen Entwicklung drohen. Werkzeugketten, statische Verifikation, Partitionierung, Speicherverwaltung, Multi-Core und modellbasierte Vorgehensweisen sind einige der betrachteten Themen, die in diesem Zusammenhang eine wesentliche Rolle spielen und allen Beteiligten, ob in

Forschung, Entwicklung oder Hochschulen, am Herzen liegen. Autonomes Fahren unter massiver Vernetzung und hochkomplexer Informationsverarbeitung mit interpretativen Entscheidungen des Fahrzeugs erfordern höchste Zuverlässigkeit und gleichzeitig transparentes und nachvollziehbares Verhalten. Inwieweit der mathematische Beweis des zeitlich korrekten und deterministischen Verhaltens bei dieser Softwarekomplexität noch gelingt, muss intensiv diskutiert werden.

Diese Interessenlage spiegelte sich auch im Spektrum und den Themen der Einreichungen zu dieser Tagung wider, die deutlich die Fragen der Sicherheit der Systeme ins Zentrum stellten. An dieser Stelle sei dem Programmkomitee gedankt, das zu den nötigen vielen Reviews bereit war, um eine solide Abdeckung bei der Begutachtung aller Beiträge zu gewährleisten. Aus der Menge der eingereichten Papiere wurden sorgfältig die besten und einschlägigsten Beiträge zur Publikation und zum Vortrag ausgewählt. Leider konnten nur weniger als 40 Prozent der Einreichungen in das Konferenzprogramm aufgenommen werden. In fünf Sessions wurden die angenommenen Beiträge dem Publikum präsentiert. Eine Paneldiskussion zum Thema „Zuverlässigkeit – Safety – Security“ schlug den Bogen um die diversen Themen. Die Schlusssession bot die Gelegenheit, Preise für den besten Beitrag und die beste Präsentation zu vergeben.

Die Tagung ist Teil einer Serie, die seit 2004 im zweijährigen Turnus veranstaltet wird. Die Tagungsbände der Serie sind Zeitzeugen eines hochinteressanten Wandels der Interessen rund um software-gestützte Funktionen im Automobil. Seit dem anfänglichen Ringen um bessere Zuverlässigkeit in den damals verbesserungsbedürftigen Systemen haben wir das Entstehen von einschlägigen Richtlinien und Standards, z. B. EN 26262, erlebt, deren Notwendigkeit 2004 noch sehr umstritten war. Seit kurzem ist zu beobachten, dass sich das Interesse der Sicherheit zuwendet, nun, da Automobile nicht mehr autarke Einheiten sind, sondern beginnen, in globale Kommunikationsnetze eingebunden zu sein. Die bislang weitgehend ungesicherten Kommunikationsprotokolle im Auto müssen mit den nötigen technischen Mitteln gegen unautorisierte Zugriffe und Beeinflussungen abgesichert werden. Weltpolitische Ereignisse haben gezeigt, dass das Eindringen in Systeme nicht nur von einigen wenigen böswilligen Hackern in Garagen betrieben wird. Sicherlich haben sie dazu beigetragen, dass Sicherheit zu einem globalen Thema geworden ist. Multi-Core-Architekturen sind eine weitere Entwicklung, die neue Sicherheits- und Zuverlässigkeitsfragen für die IT im Fahrzeug aufwirft, da sich unabhängige, nicht auf einem Mono-Prozessor eingeplante Anwendungen die Ressourcen teilen müssen, ohne sich gegenseitig unzulässig zu beeinträchtigen. Aber auch die das Fahrzeug umgebenden Prozesse, von der Diagnostik in der Werkstatt und der Wartung über die Bereitstellungsmethoden von Telefonie und Infotainment, bis hin zur Erfüllung gesetzlicher Anforderungen rund um E-Call-Dienste, müssen in die Sicherheitsanalysen aufgenommen werden und zweifelsohne Neuerungen erfahren.

Die Abendveranstaltung bot das Konferenzdinner in einem industrie-schicken aber gemütlichen Ambiente mit zusammengewürfeltem Mobiliar, losen Glühbirnen und abgeblättertem Putz an den Wänden im Kontrast zu den hohen Sicherheitsansprüchen aus den Vorträgen der Tagung.

Die fachlichen Träger der Automotive-Tagung sind die Fachgruppen Ada, ENCRESS, EZQN und FoMSESS der Gesellschaft für Informatik in den Fachbereichen "Sicherheit - Schutz und Zuverlässigkeit" und "Softwaretechnik" sowie der Fachausschuss "Embedded Software" der VDI/VDE-Gesellschaft Mess- und Automatisierungstechnik und der Förderverein Ada Deutschland e.V. Die Tagungsleitung hatten Hubert B. Keller, Karlsruher Institut für Technologie, und Peter Dencker, ETAS GmbH, inne. Die wissenschaftliche Leitung erfolgte durch Hubert B. Keller, Karlsruher Institut für Technologie, und Erhard Plödereder, Universität Stuttgart. Unser Dank geht an das restliche Organisationsteam, bestehend aus Peter Dencker, ETAS GmbH (Ausstellung), Christoph Grein (Web), Reiner Kriesten, Hochschule Karlsruhe (Tutorials), Stephanie Preissing und Mireille Pistorius, ETAS GmbH (Tagungssekretariat und -vorbereitung), Herbert Klenk, Airbus Defence and Space GmbH (Finanzen und Tagungsband) und Silke Lott, ETAS GmbH (Organisation vor Ort).

Der besondere Dank der Tagungsleitung geht an die ETAS GmbH für die großzügige Bereitstellung der Tagungsstätte und der Tagesverpflegung der Tagungsteilnehmer und an Frau Cornelia Winter, Gesellschaft für Informatik, für die zügige Bearbeitung der Proceedings. Dank auch an den Förderverein Ada Deutschland e.V. für die Übernahme der finanziellen Verantwortung für die Gesamtveranstaltung.

Karlsruhe und Stuttgart, im Frühjahr 2015

Hubert B. Keller Erhard Plödereder

Tagungsleitung

Hubert B. Keller, Karlsruher Institut für Technologie (KIT)
Peter Dencker, ETAS GmbH

Wissenschaftliche Leitung

Hubert B. Keller, Karlsruher Institut für Technologie (KIT)
Erhard Plödereder, Universität Stuttgart

Organisation

Silke Lott, ETAS GmbH (Lokale Organisation)
Herbert Klenk, Airbus Defence & Space GmbH (Finanzen, Tagungsband)
Peter Dencker, ETAS GmbH (Ausstellung)
Reiner Kriesten, Hochschule Karlsruhe (Tutorien)
Christoph Grein (Web)

Programmkomitee

Gerhard Beck, Rohde & Schwarz SIT GmbH
Manfred Broy, TU München
Stefan Bunzel, Continental
Peter Dencker, ETAS GmbH
Dirk Dickmanns, Airbus Group
Simon Fürst, BMW AG
Erwin Großpietsch, EUROMICRO
Albert Held, Daimler AG
Stefan Jähnichen, TU Berlin
Jan Jürjens, TU Dortmund
Herbert Klenk, Airbus Defence & Space GmbH
Reiner Kriesten, Hochschule Karlsruhe
Thomas Kropf, Universität Tübingen
Ulrich Lefarth, Thales Transportation Systems
Jürgen Mottok, OTH Regensburg
Francesca Saglietti, Universität Erlangen-Nürnberg
Jörn Schneider, Fachhochschule Trier
Elmar Schoch, Audi AG
Claus Stellwag, Elektrobit Automotive GmbH
Werner Stephan, DFKI
Michael Weyrich, Universität Stuttgart
Hans-Jörg Wolff, Robert Bosch GmbH
Thomas Wollinger, escrypt GmbH

Veranstalter der Automotive 2014

Fachgruppen Ada, ENCRESS, EZQN und FoMSESS der Gesellschaft für Informatik
Fachausschuss 5.11 "Embedded Software" der VDI/VDE-Ges. Mess- und Automatisierungstechnik
Förderverein Ada Deutschland e.V.

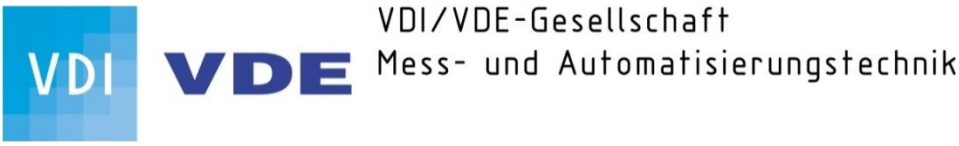
Sponsoren

FG ENCRESS

FG EZQN

FG FoMSESS

FG Ada



Universität Stuttgart

Inhaltsverzeichnis

Automotive Safety and Security Integration Challenges

Benjamin Glas[a], Carsten Gebauer[g], Jochen Hänger[h] Andreas Heyl[b], Jürgen Klarmann[c], Stefan Kriso[d], Priyamvadha Vembar[e], Philipp Wörz[f]

[dg]Bosch Center of Competence Functional Safety
[f]Chassis Systems Control, Engineering System Vehicle
[h]Corporate Sector Research and Advance Engineering
Robert Bosch GmbH
{carsten.gebauer, jochenulrich.haenger, stefan.kriso, philipp.woerz}@de.bosch.com

[b]Engineering Safety and Base System
Bosch Engineering GmbH
andreas.heyl@de.bosch.com

[ae]Bosch Center of Competence Security
[c]Software and Safety Consulting
ETAS GmbH
{benjamin.glas, juergen.klarmann, priyamvadha.vembar}@etas.com

Abstract: The ever increasing complexity of automotive vehicular systems, their connection to external networks, to the internet of things as well as their greater internal networking opens doors to hacking and malicious attacks. Security and privacy risks in modern automotive vehicular systems are well publicized by now. That violation of security could lead to safety violations – is a well-argued and accepted argument. The safety discipline has matured over decades, but the security discipline is much younger. There are arguments and rightfully so, that the security engineering process is similar to the functional safety engineering process (formalized by the norm ISO 26262) and that they could be laid side-by-side and could be performed together - but, by a different set of experts. There are moves to define a security engineering process along the lines of a functional safety engineering process for automotive vehicular systems. But, are these efforts at formalizing safety-security sufficient to produce safe and secure systems? When one sets out on this path with the idea of building safe and secure systems, one realizes that there are quite a few challenges, contradictions, dissimilarities, concerns to be addressed before safe and secure systems started coming out of production lines. The effort of this paper is to bring some such challenge areas to the notice of the community and to suggest a way forward.

Note

- The term “Functional Safety” relates to ISO 26262
- The term “Security” is used to mean Automotive Embedded Information Security
- All examples used in this paper are fictitious and do not necessarily reflect either concrete requirements or solutions.

1 Introduction

There have been calls in the past from the academic community for safety and security communities to work together [Avi04]. The need for building safety and security in automotive vehicular systems is real today, as we have started building safety-relevant systems with security requirements.

The impact of security on safety has been widely discussed and a consensus seems to have emerged. [Bur12] makes a case for the consideration of functional safety hazards arising out of malicious manipulation in addition to safety hazards arising out of systematic failures and random hardware failures by the ISO 26262 standard. [Bur12] as well as [Cze13] describe a functional safety and security engineering process. [Bur12] discusses identifying security goals in addition to safety goals in a combined functional safety-security analysis. That, security is a pre-requisite for safety and that safety could be the driver for security is an established view. There is ‘that much’ forward movement on this subject.

But is this ‘much’ really much? On the other side, there are some dissimilarities between the safety and security disciplines. The maturity-level of these disciplines is different. Automotive security is a younger discipline when compared to automotive safety. Safety has matured over decades and has had successful standardization efforts. Safety can closely align itself to legal requirements from product liability. The differing maturity levels, grey areas in law, dissimilarities in content create real challenges to realize safety and security together. The focus of this paper is to discuss these challenges and to suggest solutions or solution directions.

2 The Challenge Areas

Functional Safety discipline considers systematic and random hardware failures as hazard sources. Security considers a malicious and intelligent adversary as a threat source in addition to natural disasters and systematic failures. The unacceptable consequences for safety are loss of human life and injuries. Security as a discipline has a broader range of unacceptable consequences: human life, human security, loss of reputation, financial losses, legal violations, loss of intellectual property, damage critical infrastructure etc. These differences could be reconciled. Though, security threat sources and threats could be varied, for our purposes, we could consider ‘that’ subset of security threats that lead to safety consequences. But, there are many other differences that cannot be this easily reconciled. We focus on such challenge areas in this paper.

2.1 The Misuse Challenge

[Bur12] discusses a method for performing a joint functional safety and security analysis and illustrates the same with an example. The security analysis process begins by identifying 'Misuse cases'. The misuse cases are later analyzed for safety risks. The differences begin right here. The differences become apparent when safety and security teams discuss what 'misuse' means to each of them. The safety community believes in preventing 'foreseeable misuses' by the 'user' of the product. But, when the security community talks about misuse, they are talking about a whole range of 'negative scenarios' that could be brought about, not just by the user (who could also misuse the system, for example, by performing chip-tuning), but also by an external malicious agent who possesses the wherewithal in terms of capabilities and resources to cause not just loss of life and injuries but a greater large-scale damage. The picture below illustrates this range of possibilities.

The Misuse Challenge

		Impact on
Vehicle Owner	Activate features (unpaid, or unreleased), Use Patches or SW from internet, Manipulate or fake crash recorder data etc.	Safety, Privacy Financial
Diagnostics/ Workshops/ Installers/ Third parties	Motivate repairs, Activate (unpaid, unreleased) features, Software reflashing, Changing parameters, Unauthorized reading and clearing of crash recorder data, Refurbishing Components, Resetting error counter while selling or billing new Components etc.	Safety, Privacy, Customer Trust, Financial
Tuner	Tuning Services, Build your own vehicle, Mix Components (ex. ECU's) from different vehicles etc.	Safety, Customer Trust, Financial
Hacker	Remote function invocation or rejection of function Requests (ex. remote braking or rejection of braking requests), Denial-of-Service, Manipulation or shut down of safety monitoring, Actuator manipulation, Steal personal data, Build user profiles, Execute unauthorized commands, Malware installation (bubbling CAN) etc.	Safety, Customer Trust, Critical infrastructure damage, Privacy
Terrorist	Through Remote Control, Use Vehicle as a Weapon, Cause accidents on a large scale, Bring down critical infrastructure such as the traffic system	Safety, Critical infrastucture damage
Counterfeiter/ Competition	Steal Intellectual Property, Make unauthorized copies of SW, HW	Intellectual Property, Competitiveness, Financial
Criminal	Deactivation of safety functionality, Deactivation of Brakes etc.,	Safety

Figure 1: The Misuse Challenge

These acts would perhaps qualify for a carefully hatched 'sabotage'; not a foreseeable misuse really, and hence would amount to no penalties from the product liability legislation for non-consideration of the threat in product development. What sort of security 'Misuse' would be a misuse that must be prevented, on which efforts need to be invested? Do we want to make distinctions between acts that could be easily performed by a 15-year old armed with a computer, a set of downloaded tools and an instruction manual from the ubiquitous 'You Tube'? Do we accept the risks presented by a well-versed hacker, a cyber-criminal or a terrorist as sabotage that need not be prevented or detected? Where should the line for risk acceptability be drawn? What is the legal opinion on this? Do we need consensus and policies here, at least, for the automotive world?

Our view on the subject is as follows: The EU directive 2001/95/EG, Article 2 gives the following definition: "'safe product' shall mean any product which, under normal or

reasonably foreseeable conditions of use [...] does not present any risk or only the minimum risks compatible with the product's use, considered to be acceptable and consistent with a high level of protection for the safety and health of persons [...]" [EG01]. Safety activities focus on normal and reasonably foreseeable conditions of use. In the German product safety act ("Produktsicherheitsgesetz") this is named "beabsichtigte und vorhersehbare Verwendung" ("intended and foreseeable conditions of use") [PSG11]. This is also the intention of ISO 26262: When performing a hazard analysis and risk assessment (H&R), intended use (normal conditions of use) inclusive reasonably foreseeable use is considered (ISO 26262-3, 7.4.3.7, NOTE 2). For example, when performing a H&R it is "intended use" that someone does not drive faster than a given speed limit, e.g. 50 km/h. But it is reasonably foreseeable that someone drives "a little bit faster" than this given limit, e.g. 60 km/h. But it is not reasonably foreseeable that he drives much faster, e.g. 200 km/h. But it is difficult to define a clear borderline between "foreseeable" and "not foreseeable" here.

According to ISO26262 safety is defined as the "absence of unreasonable risk" (ISO 26262-1, 1.103). The risk coming from a system is the "combination of the probability of occurrence of harm and the severity of that harm" (ISO 26262-1, 1.99). Possible sources of risks are malfunctions coming from random hardware faults during operation of the system as well as systematic faults during development of the system. Both these kinds of faults are addressed in ISO 26262, "functional safety of road vehicles".

Another source of risk could be a malfunction coming from the manipulation of the system, e.g. a hacker attack, which could provoke safety-critical behaviour of the vehicle (independent thereof whether this is intended by the hacker or if it is only an unintended side effect). If the probability of such a manipulation (in combination with its severity) is so high that it leads to a "unreasonable risk" level, measures for risk reduction are necessary. Basically, measures against manipulation are addressed in the area of "automotive security", not in the area of "functional safety" – but they have to be considered as "safety measures" in the sense of "risk reduction measures". For example, if an engine management system is not hardened against hacker attacks, such an attack could lead to safety-critical behaviour of the vehicle (e.g. braking system not being able to deal with a higher engine power).

Even here, it is difficult to define clearly the probability of manipulations and with that, the necessary risk reduction measures. Additionally, this cannot be expected to remain static over time: the probability is expected to rise, which would lead to an increase of the risk level from "reasonable" towards "unreasonable", which would make (more) risk reduction measures necessary. It is the task of field monitoring process to observe this development. If automotive security is a measure for risk reduction to ensure safety, this leads to the necessity to coordinate the safety and security activities.

2.2 The Risk Assessment Challenge

Going further, [Bur12] presents a way of performing a risk analysis. They present an approach in which safety risks are separately assessed and security risks are separately assessed using their individual methods, but as a part of H&R. But, does this kind of risk

analysis produce the right picture? The picture below displays the safety risk model along with how an attacker could fit himself into it.

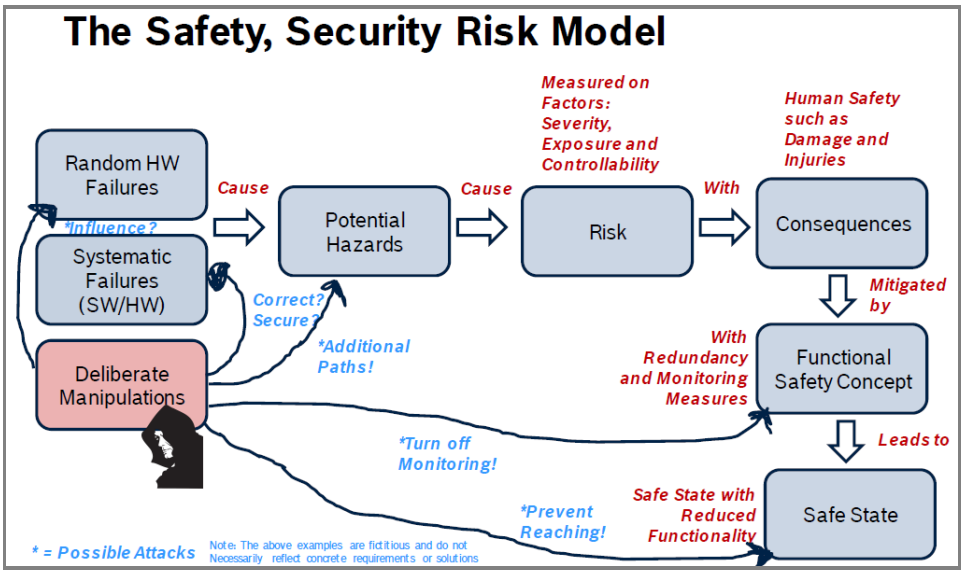


Figure 2: The Safety, Security Risk Model

The safety risk model is oriented towards dealing with risks that arise due to natural or random causes in nature. But, the introduction of an 'intelligent' adversary alters the risk landscape significantly. This makes the 'Hazard and Risk Analysis (H&R)' on the safety side and 'Threat and Risk Analysis(T&R)' on the security side, inherently incompatible.

The H&R tries to find and classify the risk, which is related to an item. The risk classification is documented via the "ASIL" (Automotive Safety Integrity Level). The classification is based on three parameters: (a) The potential severity of an hazardous event (severity), (b) probability of an operational situation (exposure) in which this event might happen, and the (c) ability of traffic participants to control the event (controllability). Within the classification, external safety measures can be considered to be affecting the exposure, severity or controllability to reduce the ASIL.

As a result of the H&R, safety goals with ASIL shall be defined. If similar safety goals are determined, these may be combined into one safety goal. In this case, the highest ASIL shall be assigned to the combined safety goal.

The picture below presents a high-level comparision of the safety and security risk models.

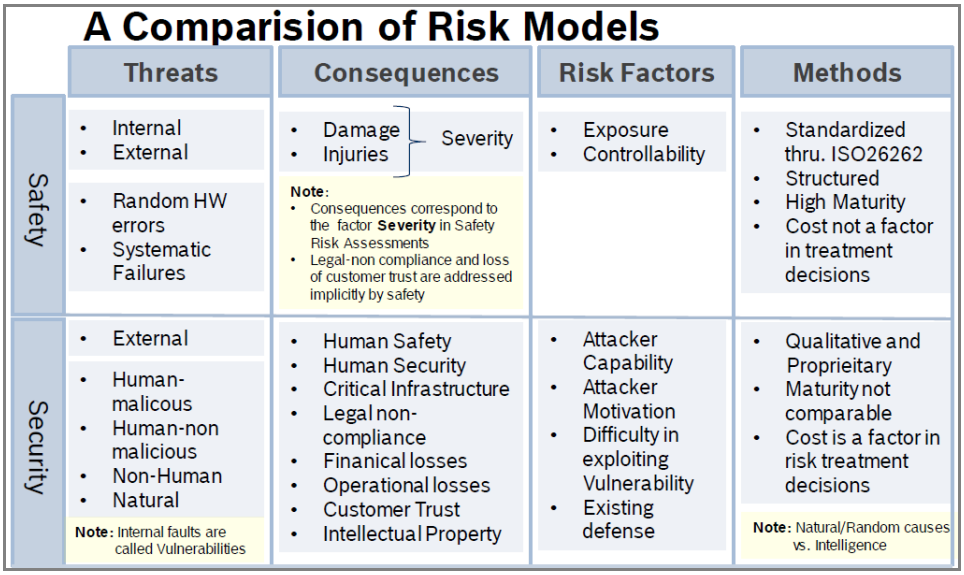


Figure 3: A Comparision of Safety and Security Risk Models

“Consequence” on the Security side, and “Severity“ on the safety side is the common parameter for security and safety. The probability-related parameters nevertheless are quite different, adressing the attack potential of the hacker and the attack potential the system is able to withstand: available time for an attack, required specialist expertise, required system knowledge, window of opportunity to access the target of attack, required equipment etc.

Safety is a common asset for both the domains. Both try to assess the safety-related risk which might be a result of random or systematic faults on the one side or deliberate manipulation on the other, to derive the required “strength” of the corresponding countermeasures. Both get to the risk classification by applying the mathematical definition: risk = severity X the probability of an event. Therefore, it seems feasible to base ones T&R on the results of the H&R to save time [Bur12, Cze13]. In the following chapter, we show by counter-arguments why this could lead to misunderstandings and why the commonalities seem to be limited to the rough process steps and to the basic definitions.

The first point of criticism adresses the classification parameters, which can not be used on both sides. For example one can construct many E1-situations, in which a malfunction is hard to control (C2 or C3) and leads to severe or lethal injuries (S2 or S3). But due to the fact, that these situations are so rare, ISO 26262 will classify the corresponding malfunction with a low ASIL, e.g. QM or ASIL A. For a hacker this small likelihood is not that relevant, since he can either wait for such a situation or even provoke it before triggering the corresponding attack. At an advanced level, he can even have a “sleeping” function waiting for a specific situation by evaluating vehicle data and triggering an attack automatically.

Additionally, the hacker might be able to also influence the controllability. This can be achieved by performing a parallel attack to distract the driver in the moment of the main attack, e.g. by impairing his field of vision by flushing wiper fluid [Kos10] or irritating him by increasing the radio volume.

Composite attacks on available and considered external measures (e.g. ESP being external of the item "powertrain") could also be used to impair their capability to mitigate a hazardous malfunction of the system. Therefore the focus of the H&R on an item seems to be insufficient for theT&R, which has to widen its scope to the whole vehicle or even the environment.

The conclusion is, that the "probabilistic" filter of the H&R makes it rather impossible to come up with a simple mapping to transfer the results of a H&R into a T&R. Rather, all situations and parameters have to be assessed again under a security point of view ("unlikely vs. unattractive"), applying security-specific methods and checklists. Since, within the H&R, typically implicit assumptions are made, which might lead to a reduction or grouping of situations, even the situation catalogue resulting out of a H&R might be incomplete for the security considerations in the T&R. Probably only the definitions of the terms and the various classes of "severity" and "controllability", that are already defined in ISO 26262, could be a candidate for a common use.

But even if these methods (H&R and T&R) don't go together very well, the experts of these two fields might and should. Hence it is recommended, that experts of both fields exchange and discuss their risk analysis results: (a) to check them for completeness, at least regarding the asset "safety", (b) to become aware of the subsequent functional and technical requirements for each domain, that might affect each other later on in development, and (c) to gain a mutual understanding of each others field and their special needs, approaches and contradictions.

2.3 The Solution Space Challenges

The issues deepen as we move into the area of finding solutions that are acceptable to both the safety and the security communities.

Safety and security share the common goal of protecting integrity – meaning, the correct and intended functionality of the system against failures, errors, and mainly external influences. Security usually has more goals and other assets to protect, but in this area one finds the greatest overlap. Towards meeting this objective, both use a similar set of mechanisms, methods and resources, e.g. adding redundancy, monitoring, testing, and verification. Naturally, if applied independently, this often results in conflicts since both disciplines need access to the same set of limited resources: bandwidth of communication, storage space, processing time, access to flash and RAM, and others. Before we examine approaches to generate synergies, let us consider two examples of actual or possible conflicts.

2.3.1 Example 1: Failure Analysis of Issues during Product Usage

Field monitoring of a product is required by several laws (e.g. w.r.t product liability), standards (e.g. ISO 26262-7), as well as internal and external regulations.

Typical points of analysis from a safety point of view include:

- Flash analysis by identification of software version, checking the integrity of flash hardware and content and even dumping the flash content
- RAM analysis by checking hardware and content, and also modifying RAM content during runtime for debug reasons
- Check of EEPROM content (e.g. failure memory)
  Check of ECC (error correcting code) information of microcontroller registers

In general, these analyses require access to the microcontroller in debug mode.
Typical stakeholders for these activities are

- Tier 1 Developer (System, HW and SW)
- Tier 1 Manufacturing and Quality Assurance
- OEM Manufacturing and Quality Assurance

From security point of view, this is at least partially a contradiction to the security goals and interests, because the security interests would read like the following:

- Protect flash content against manipulation and re-engineering (protect intellectual property)
- Protect RAM content against manipulation, restrict access during runtime (in debug mode) used for system analysis to authenticated and authorized users
- Protect EEPROM content to prevent misuse cases like:
  - modification of failure entries in order to shift responsibility for an event (from driver to system)
  - read access to confidential data, for example, as a part of re-engineering
  - read access to user specific data, which might violate privacy rights of the user.
  - write access to system configuration data to activate unpaid functionalities (which could lead to loss of revenues)

It can be seen that, while the basic goal of preserving ‘Integrity’ remains the same across the disciplines, the methods employed by safety and security may conflict in the actual application. One good example is the ease of access to system resources. While safety implicitly expects that access should be easily possible for a quick and easy analysis, security would restrict access as strictly as possible via authentication and authorization mechanisms. In many cases, this can be resolved by granting authorization specifically to valid safety-monitoring entities at the cost of added (management) overhead. Nevertheless a trade-off needs to be found between allowing access for legitimate and authorized safety features while preventing an attacker from using this access to violate security goals. In order to distinguish a valid safety-mechanism from an attack, a co-design for safety and security mechanisms is necessary with need for compromise on both sides. Some safety mechanisms need re-evaluation, as to whether they can be used in presence of security needs.

2.3.2 Example 2: RAM Test

As part of a standard safety concept, RAM tests are required. Usually, these RAM tests include (temporary) modifications of RAM content by writing test patterns into selected RAM sections. For other users of the RAM such as bus masters, it is impossible to distinguish between real RAM data and test patterns. This could lead to unintended or even safety-critical behavior of the system. In order to prevent usage of these RAM test patterns, usually all bus masters are stalled during the RAM test execution. This is a well-established design pattern from the safety perspective.

But, from a security point of view, one of the main security requirements is to prevent stalling of main security (hardware) components like Hardware Security Modules (HSM). Without further measures this would lead to the possible misinterpretation of RAM test patterns as valid code or data by HSM and therefore random HSM behaviour.

Both examples do not show fundamental contradiction in requirements. Nevertheless, they show that :

- Current well-established design patterns are not feasible anymore when security requirements are added
- Mutual dependencies have to be analyzed
- Common design for safety and security aspects is necessary in future

Current automotive reality is in the situation, that safety is an integral element of system design, while security features are new to many automotive subsystems. Integration of both sets of requirements needs compromises or trade-offs on both sides. An understanding on the side of security that the established safety mechanisms are not easy to change, and an acceptance on the side of safety that security is not just a “plug-in” that can be applied independently and on top of the legacy safety system, but needs integration and adaptation to the existing and proven safety approaches.

3 Synergies between Safety and Security

3.1 The CRC and the MAC

One area where safety and security goals coincide is in the area of integrity protection for data – both in transit and in storage. In order not to use corrupt or manipulated data, integrity has to be protected or at least loss of integrity has to be detected. The difference between the domains is the perceived threat for the integrity. While (functional) safety aims at protection against systematic errors and random errors caused by malfunction or unintended interference, security additionally wants to protect against targeted, intended and possibly malicious manipulation. This leads to different approaches for protection, which nevertheless have some common properties. The general idea is always to add redundancy to the data, so that a consistency check detects deviation from the original data. The amount (number of bits) of redundancy correlates with the probability of detecting an error, the more redundancy invested, the more errors can be detected.

Differences are in the approaches to generate this redundancy. In the following section, we take a look at two prominent representatives.

Safety: The Cyclic Redundancy Check (CRC)

Cyclic redundancy check denotes a class of mechanisms to detect and partly also to correct randomly distributed errors using additional redundant data generated by binary polynomial division. The polynomial can be chosen to adapt to different requirements. Basic assumption is usually a binomial distribution of single bit errors, which implies e.g. that the probability of an error pattern decreases with the number of changed bits and particularly single bit errors are more likely than multi bit errors. Most CRCs in use guarantee detection of errors of some classes (e.g. single bit errors or all errors up to a certain number of changed bits) and even offer correction for a subset of these classes. CRCs are easy to generate and evaluate both in hardware and software and need no secret information to be computed. They are widely used, e.g. in automotive communication (e.g. CAN bus protocol).

Security: The Message Authentication Codes (MACs)

Message Authentication Codes are a class of mechanisms using basic cryptographic primitives like keyed cryptographic hash functions or block ciphers to create integrity tags that can only be created and evaluated with the knowledge of a secret key. They belong to the cryptographic class of symmetric primitives, meaning that the key used for generation of a MAC is the same as the one needed for verification.

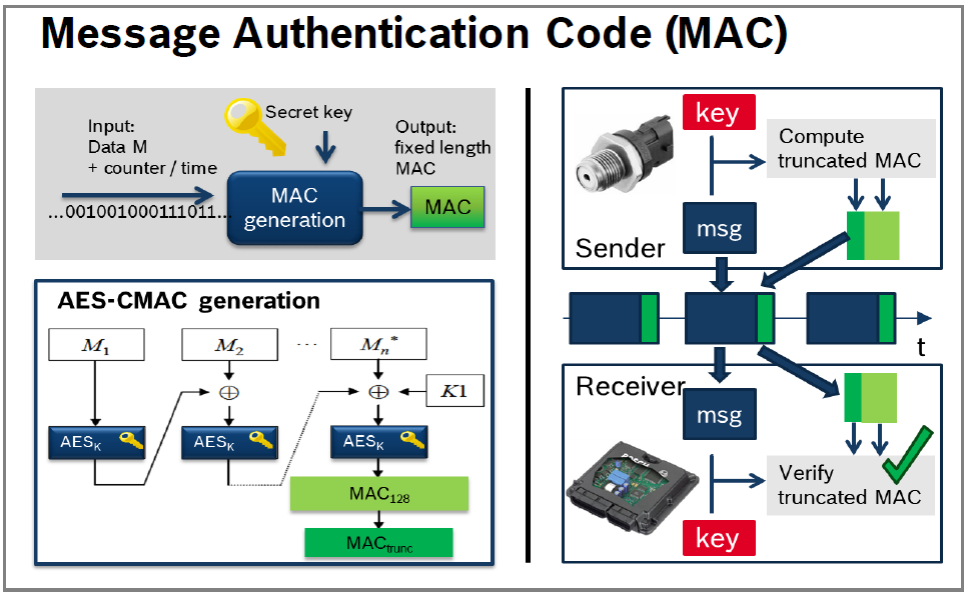


Figure 4: The Message Authentication Code(MAC), an Introduction

In contrast to CRCs, MACs have different basic design criteria, since the basic goal is authentication of data and/or originator of the message. This leads to the desired property, that is has to be infeasible for an adversary to create a valid MAC for a

message without knowing the key. This has to be guaranteed even if the abstract adversary has access to an arbitrary number of message-MAC-pairs. So, even if for any number of similar messages the MAC is known, “derivation” of the MAC tag for a different message must not be feasible. This also leads to the properties, that for similar messages, the MACs must not be similar, at least statistically, and that from a MAC it should be infeasible to extract information about the message.

Looking at cryptographic MACs, it can therefore be usually assumed, that errors/manipulations in the message are detected with a constant probability of $2^{-(\text{length of MAC})}$ for all errors, independent of the error class (e.g. hamming distance). No guarantees can be given for specific types of errors. Also no correction is possible. While mathematically this is the best error detection possible with the redundancy invested assuming a uniform distribution of errors, this is not the best detection assuming a binomial error distribution.

Algorithmically, the receiver cannot distinguish between a random error and a malicious manipulation of data. Therefore the MAC aims at detecting any deviation from the original message which includes and goes beyond all error causes looked at from a safety point of view. Therefore a MAC could in principle replace a CRC for error detection purposes in resource constrained in systems in order to enhance the functionality to authenticity protection, if the uniform detection probability is acceptable from a safety point of view.

3.2 Virtualization for Embedded Systems

Virtualization in computing refers to the act of creating virtual computers on one physical computer. Here, we refer to such virtualization technologies only, where the hypervisor runs directly on the hardware. These are most accepted for embedded systems and provide features that support both safety and security.

The main features of virtualization that are of interest to us are:

- Strong isolation between virtual computers: Two virtual computers do not know one another and run like independent physical computers. This property is provided by a hypervisor which provides a strong separation between both virtual computers
- Communication: A communication channel between both virtual computers may be provided and controlled by the hypervisor (comparable to a data bus between physical computers).
- Hypervisors for embedded systems are typically very small and comprise only some thousand lines of code. Experts say that they come with a “small trusted code base”.
- Cost benefit: Introduction of virtual computers does not increase serial production costs for physical computer and do not increase further costs in the vehicle e.g. for harness or for mounting.

The picture depicts hypervisor uses for safety, security and both safety and security.

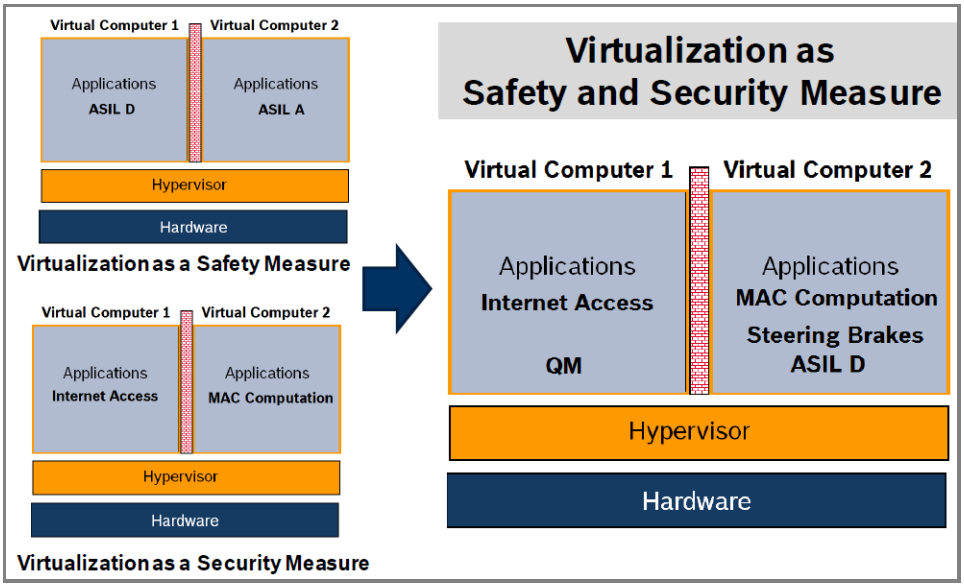


Figure 5:Virtualization as Safety and Security Measure

Virtualization as a Safety Measure: The aim of functional safety standardizations like ISO 26262 is to minimize hazards and risks to a reasonably low level. As hazards and risks are of different nature, it is customary to encounter different criticality levels (ASIL A to ASIL D) during system design. ISO26262 demands development according highest ASIL for whole software as long as sufficient freedom from interference cannot be guaranteed between applications with different ASIL. Development according to highest ASIL is often not possible when software from different sources shall run on one physical computer. The strong isolation feature of virtualization guarantees required freedom from interferences between applications of different ASIL. Thus, each virtual computer may run software with an ASIL of its own.

Virtualization as a Security Measure: Virtualization comes with features which are a great help for designing secure systems. Due to strong separation of virtual computers, each one may run independent of the others comparable to applications running on individual physical computers. Thus a security attack on one virtual computer does not affect the others. In addition, each virtual computer may be stopped or restarted independently. That allows for a design where applications prone to security attacks (e.g. applications with internet connectivity) run in one virtual computer while applications with safety or security critical tasks (e.g. steering brakes or MAC computation) run on another virtual computer. Such a design is further supported by the small trusted code base that hypervisors typically come with. Typically, small code of the hypervisor can get hardened to ensure the strong isolation between the virtual computers.

Virtualization as Safety and Security Measure: Virtualization may serve as a measure for both safety and security on one physical computer as shown in the figure above. The figure shows one physical computer on which two virtual computers run applications of different ASIL and different security criticality.

3.3 Making Programs Run Time Error Free

A run-time error is an error detected after or during the execution of a program in contrast to the errors detected during compile-time. Run-time errors such as invalid usage of pointers and arrays, invalid ranges and overflows, division by zero, invalid function calls (wrong parameters, recursion), or use of uninitialized variables are often hard to find by running test cases. Run-time errors may lead to arbitrary failures.

Run-time errors are classified as systematic faults in safety and may lead to safety relevant malfunctions. ISO 26262 (book 6, table 1) suggests the use of language subsets to exclude language constructs which could result in unhandled runtime errors. However, in the most used embedded programming language C, runtime errors can be caused by *basic operations and thus absence cannot be ensured by exclusion by a language subset.*

According [Wal09.] most security attacks are input or output related. Today's security vulnerabilities lie rarely in cryptographic algorithms or protocols. Almost always these security vulnerabilities are implementation related. Lists like the "Recent Vulnerability Notes" [Cer40] demonstrate that very well.

Thus, the guarantee of absence of runtime errors helps safety and security significantly.

Sound abstract interpretation may guarantee absence of all runtime errors. The aim of abstract interpretation is to obtain information about the behavior of programs, by abstracting parts of the program and the instruction retraces step by step. In abstract interpretation one focuses on some aspects of the execution of the instructions, abstracting out some details, giving an approximation of the program semantics, which is sufficient for the desired purpose. "Sound" means all runtime errors are found. Thus we don't refer to methods and tools here which are unsound, which means, they may find only a portion of all runtime errors. Abstract interpretation consists of considering *abstract semantics*, which is a superset of the concrete program semantics. The abstract semantics cover all possible cases. If the abstract semantics are ok then, so also the concrete semantics [Abs14]. Sound abstract interpretation is able to detect all kinds of runtime errors in C source code.

3.4 Combining Virtualization and Runtime Error Removal

The usage of the hypervisor in the approach depicted in the previous chapters is based on the ability to create a strong separation between the virtual computers. The strong separation is realized by a typically small trusted code base. But even that small trusted code base may contain runtime errors which may endanger safety and security goals.

Hardening of hypervisor code by sound abstract interpretation is a clear possibility.The application of sound abstract interpretation for a hypervisor has the advantage of being cost-efficient due to its small code base. Together, these measures offer a great potential in being used in applications requiring both safety and security.

4 The Misconceptions

Until now, we discussed challenges and synergies. There also exist misconceptions that arise from carrying the 'Safety Paradigm' into Security. A prominent example of this is with regard to the 'Safe State'. Such questions are asked: Upon detection of a hazardous situation, safe state would be to 'Open the doors' to let the passengers out, but would 'Secure State' be to shut the doors to keep the intruders out? It would perhaps be necessary (in hypothetical situations) for a safety-relevant system to write to some areas of memory, modify the internal state to bring the system to safe state in emergency situations. Is security against this? Will it prevent it? Are the requirements contradictory?

The answer is: There is no such thing called a 'Secure State' defined in the domain of security. Through information security, we intend to protect information assets from loss of confidentiality, integrity and availability against an unauthorized intruder. Taking this idea, a step further, even if there were to be a 'Secure State', it would be an integral 'Safe State' in which the confidentiality of sensitive information is appropriately protected and the resources needed to attain a safe state and the resources needed to persist in the safe state are available and are protected from malicious denial-of-service.

In general, security takes on those goals that are expected of it by the system. Hence what security does is dependent upon the goals of the system in question. Security per se, does not possess independent goals of its own. This holds for any safety-relevant system.

5 Methodological Proximity

There is methodological proximity between the two disciplines. At a high level of abstraction, the engineering processes appear similar. But, there is more to the story.

The risk models of safety and security are entirely different in nature. Safety Risk Model is based on random events in nature, whereas, security considers risks due to malicious human intelligence. Hence the risk analysis, assessment approaches differ. The risk models cannot be superimposed on one another, nor can there be uniform approaches for assessment though both protect the same asset 'safety'. Hence, Safety and Security require a Risk Co-Analysis approach.

At the level of countermeasures, safety takes a control system view and a detective approach, but the focus of security would still remain on preventative measures (though detective and recovery measures do have their place). There are challenges and potential synergies in safety and security measures, that require a shift in the thinking on the

safety side. For the security side, a word of caution is in order. Projects with safety relevance are not green-field projects. There is work in integration which needs time and interaction. Hence, we propose safety+security Co-Design to achieve the integration.

At lower levels of abstraction, such as coding, security needs to go a step further from safety. Safety needs correctness, but security needs correctness and some thing more to be secure.

In the area of verification and validation, security test cases can build upon existing test-case development methods for software testing such as equivalence partitioning, boundary value analysis, multiple condition coverage etc., with additional test cases for testing the preservation of Confidentiality, Integrity and Availability (CIA) at all levels.

6 Conclusion

Safety-Security integration is an important integration for automotive vehicular systems. The processes look similar and could be performed together. But, attention to detail is essential. The safety and security communities need to engage at a deeper level than today to make the integration happen. Any process for safety and security (isolated or combined) needs to be aware of the inherent differences and commonalities between the two disciplines. A security process needs to be 'Safety Aware' and vice-a-versa. Hence, safety and security processes performed by different experts - is clearly not the way forward. Separate safety and security standardizations by themselves don't help either. Finding vocabulary, developing tools that could be used by both communities in a similar manner, resolving or at least developing similar understanding of grey areas in law, looking for mechanisms, architectural building blocks, design patterns that could work for both the domains holds the key to bringing forth a meaningful, working and 'Safe' integration

References

[Avi04] Avizienis A, Laprie J-C, Randell B, Lanwehr C, "Basic Concepts and taxonomy of Dependable and Secure Computing", IEEE Transactions on independent and Secure Computing". January-March 2004.

[Bur12] Simon Burton, Juergen Likkei, Priyamvadha Vembar, Marko Wolf, "Automotive Functional Safety = Safety + Security", In: 1st International Conference on Security of Internet of Things (SecurIT 2012), Kerala, India

[Cze13] Czerny, B., "System Security and System Safety Engineering: Differences and Similarities and a System Security Engineering Process Based on the ISO 26262 Process Framework," *SAE Int. J. Passeng. Cars – Electron. Electr.Syst.* 6(1):2013, doi:10.4271/2013-01-1419

[EG01] Directive 2001/95/EC of the European parliament and of the council of 3rd December 2001 on general product safety

[PSG11] Act on making products available on the market (Product Safety Act) / Gesetz über die Bereitstellung von Produkten auf dem Markt (Produktsicherheitsgesetz, ProdSG), 8. November 2011 (BGBl. I S. 2178, 2179; 2012 I S. 131)

[Kos10] Karl Koscher et al. “Experimental Security Analysis of a Modern Automobile”, University of Washington Seattle, University of California San Diego, http://www.autosec.org/pubs/cars-oakland2010.pdf, 2010

[Wal09.] Walter Kriha, Roland Schmitz, “Sichere Systeme, Konzepte, Architekturen und Frameworks”, Springer Verlag, 2009

[Cer14] Vulnerability Notes Database, http://www.kb.cert.org/vuls/, 29.10.2014

[Abs14]“Abstract Interpretation in a Nutshell”, http://www.di.ens.fr/~cousot/AI/IntroAbsInt.html , 29.10.2014

[ISO11] International Organization for Standardization, ISO 26262, “Road Vehicles – Functional Safety”, International Standard

Integrationslösung zur Absicherung eines realen Radarsensors im Systemverbund mit der Hardware-in-the-Loop Testtechnologie

Marco Weiskopf, Christoph Wohlfahrt

RD/EDT
Daimler AG
HPC G025-BB
71059 Sindelfingen
marco.weiskopf@daimler.com
christoph.wohlfahrt@daimler.com

Dr. Albrecht Schmidt

Institute for Visualization and
Interactive Systems (VIS)
Universität Stuttgart
70569 Stuttgart
albrecht.schmidt@vis.uni-stuttgart.de

Abstract: Aufgrund der steigenden Komplexität, Anzahl und Vernetzung der Fahrerassistenzsysteme wächst die Notwendigkeit einer ausreichenden Absicherung in Bezug auf Zuverlässigkeit und Sicherheit. Dabei kommt die Hardware-in-the-Loop (HiL) Testtechnologie zum Einsatz, die automatisierte Tests in Echtzeit mit der realen Steuergeräte-Hardware ermöglicht. Bei der Durchführung von Systemtests für Fahrerassistenzsysteme am HiL-Prüfstand wird es immer wichtiger alle Komponenten wie z.B. Radarsensoren zu integrieren um eine möglichst realitätsnahe Testaussage zu bekommen. Viele Steuergeräte erfassen die Umgebung und benötigen ein gemeinsames Fahrzeugumfeld. Hierfür wird eine realistische 3D-Animation (Virtuelle Welt) verwendet. Dieser Beitrag zeigt zum einen, wie aus der virtuellen Welt relevante Daten für den Radarsensor gewonnen werden können. Dabei spielen Materialeigenschaften, Berechnung von Reflexionen, realitätsnahe Generierung, Interpolation von Detektionen und Echtzeitfähigkeit eine bedeutende Rolle. Zum anderen beschreibt dieser Beitrag eine echtzeitfähige Lösung zur Einspeisung der relevanten Daten in den Radarsensor.

1 Einleitung

Eine lange monotone Autobahnfahrt kann dazu führen, dass die Konzentrationsfähigkeit des Fahrers sinkt. Hierbei unterstützen Fahrerassistenzsysteme, indem sie kontinuierlich und unermüdlich die Fahrzeugumgebung nach Gefahrensituationen analysieren und falls erforderlich in die Fahrzeugdynamik eingreifen. Dabei kommen hochkomplexe Erkennungs- und Klassifikationsalgorithmen zum Einsatz, die verteilt auf verschiedenen Steuergeräten die Umgebung und den Verkehr beobachten und falls erforderlich die Fahrdynamik beeinflussen (siehe Abbildung 1). Diese Algorithmen dienen für Fahrerassistenzsysteme wie z.B. Abstandregeltempomat, Notbremsassistent und Ausweichassistent.

In Bezug auf die Sicherheit sind Eingriffe in die Fahrdynamik kritisch zu bewerten. Deswegen wird bei Daimler zur Absicherung dieser softwarebasierten

Fahrzeugfunktionen eine konkrete Teststrategie festgelegt. Ein wesentlicher und bedeutender Bestandteil dieser Teststrategie sind HiL-Tests auf Komponenten-, System- und Fahrzeugebene. Diese ermöglichen bereits in frühen Entwicklungsphasen ein reproduzierbares Testen im Labor

Abbildung 1: Fußgängererkennung mittels Radar (Halbkreise) und Kamera(Dreieck)

Um diesen Vorteil der HiL-Test zu nutzen, ist es notwendig alle am zu testenden System beteiligten Komponenten an das HiL-Testsystem anzubinden. Die Anbindung kann dabei auf zwei Arten erfolgen. Es wird das reale Steuergerät verwendet oder eine Restbussimulation. Erstrebenswert ist es, immer die realen Steuergeräte zu verwenden um eine möglichst realitätsnahe Testaussage zu bekommen. Momentan ist es durch die Arbeit von [WWS10] möglich, die hier betrachteten Fahrerassistenzsysteme auf der System-Teststufe mit einer Stereo-Kamera als reales Steuergerät abzusichern. Dazu wurden sämtliche Sensoren und Aktoren wie Stereokamera, Fahrerassistenzsteuergerät, Radarsensor, Bremse und Lenkung in das HiL-Testsystem integriert. Die Anbindung des Radarsensors erfolgte dabei über eine Restbussimulation, welche durch diesen Beitrag durch die Anbindung des realen Radar-Steuergerätes ersetzt werden soll. Für die umgebungserfassenden Sensoren wird ein zentrales Simulationsmodell (virtuelle Welt) verwendet, zur Bereitstellung einer gemeinsamen Umgebungsbasis. Über dieses Simulationsmodell können komplexe Verkehrssituationen in Echtzeit simuliert und z.B. für die Stereo-Kamera als Videopfad zur Verfügung gestellt werden.

In diesem Beitrag wird anhand einer neuartigen Integrationsstrategie kurz veranschaulicht, wie einem Radarsteuergerät ebenfalls Daten aus dem zentralen Simulationsmodell bereitgestellt werden.

2 Verwandte Arbeiten

Dieser Abschnitt zeigt bereits vorhandene Ansätze zur Simulation von Radardetektionen auf und gibt eine Einordnung in den Kontext des vorliegenden Beitrags. Die Idee zur Simulation von Radarstrahlen, dass aus der Computergrafik bekannte Ray Tracing Verfahren von [Ka86] zu verwenden, wurde erstmals von [Ba96] vorgestellt. Durch den hohen Berechnungsaufwand auf der Central Processing Unit (CPU) eignete sich diese

Simulation nicht für den Praxiseinsatz. Um dennoch eine echtzeitfähige Radarsimulation zu Entwicklungs- und Testzwecken verwenden zu können, entwickelte [BB06] ein Verfahren zur Erzeugung von realistischen Radarziellisten. Während der Entwicklung analysierte der Autor aufgezeichnete Radarziellisten eines Radarsensors im Auto und ermittelte Punkte, die mit hoher Wahrscheinlichkeit die Radarsignale zurück reflektieren. Basierend auf den ermittelten Punkten definierte er ein Objektmodell mit Reflexionspunkten für ein Fahrzeug. Mittels einer spezifischen Simulationsumgebung erzeugte er mit Hilfe des Objektmodells eine ideale Zielliste, die durch künstliches Rauschen modifiziert wird. Ziel dieser Arbeit ist es, realistische Radarziellisten zur Unterstützung der Entwicklung von Verarbeitungsalgorithmen für Radarziellisten zu erzeugen.

Der Nachteil dieses Verfahrens liegt in der Erweiterung des Objekt-Repertoires. Denn für jedes weitere Objekt (wie z.B. Häuser, Motoräder, Fahrräder, Fußgänger usw.) muss ein spezifisches Objektmodell erzeugt werden. Für dieses Objektmodell ist wiederum eine aufwendige Analyse von aufgezeichneten Radarziellisten notwendig.

Zur Verwendung eines generischen Verfahrens für alle Objekte stellte [PLK08] einen Ansatz vor, indem er eine shader-basierte Simulation von Radardetektionen verwendet. [Sh12] optimierte die shader-basierte Simulation um diese für einen Software-in-the-loop (SIL) Testsystem einzusetzen. Für eine shader-basierte Simulation wird die Möglichkeit, eigene Shader-Programme in die OpenGL Rendering Pipeline einzuspeisen, ausgenutzt. Dadurch wird die Berechnung von der CPU auf die Graphics Processing Unit (GPU) verlagert und damit eine schnellere Berechnung ermöglicht. [Sh12] demonstriert, dass mit diesem Ansatz eine Berechnung der Radardetektionen in Echtzeit möglich ist. Die Möglichkeit, dass Radarstrahlen mehrfach reflektiert werden können ist in [Sh12] nicht berücksichtigt worden.

Um dies zu realisieren, wird wieder auf das eingangs erwähnte Ray Tracing Verfahren zurückgegriffen. In [Pa10] wird gezeigt, dass es möglich ist, mittels des Frameworks OPTIX, Ray Tracing in Echtzeit zu berechnen. Dabei wird die schnelle parallelisierte Berechnung auf der GPU für das Ray Tracing ausgenutzt.

Optix wird für die Generierung der Radar Detektionen innerhalb der hier betrachteten 3D Animation eingesetzt.

3 Bestehendes HiL-Testsystem für Fahrerassistenzsysteme

Zur Systemabsicherung von Fahrerassistenzsystemen existiert bereits ein HiL-Testsystem, dass die Stereokamera als reale Komponente integriert. Die Radarsensoren werden durch ein Modell simuliert [WWS10].

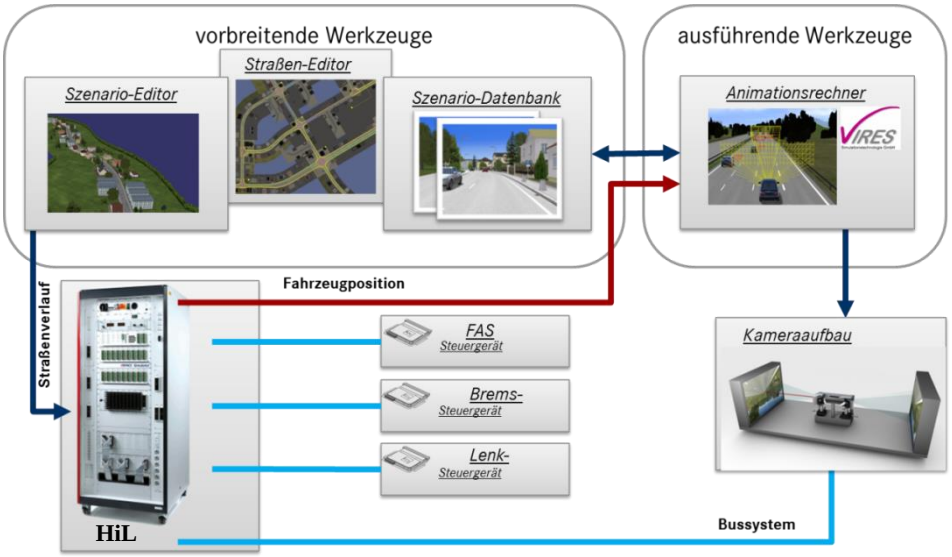


Abbildung 2: Architektur des bestehenden HiL-Testsystems für Fahrerassitenzsysteme

Für das bestehende HiL Testsystem für Fahrerassistenzsysteme definierte [WWS10] die in Abbildung 2 veranschaulichte Architektur. Das Ego-Fahrzeug wird im Echtzeitrechner des HiL-Simulators mit einer detaillierten Fahrdynamiksimulation simuliert. Über die Bus- und IO-Schnittstellen stellt der HiL-Simulator die Signale den angebundenen Steuergeräten zur Verfügung. Bei dieser Architektur wurde das klassische HiL-System durch einen Animations- und Simulationsrechner erweitert, welcher die Video-Daten für die am System beteiligte Stereo-Kamera generiert. Mittels der vorbereitenden Werkzeuge im Animationsrechner können verschiedene Szenarien definiert und in der Simulation ausgeführt werden. Über ein Netzwerkprotokoll wird der HiL-Simulator und der Animationsrechner synchronisiert.

Damit alle beteiligten Komponenten des Fahrerassistenzsystems am HiL möglichst realitätsnah abgesichert werden können, müssen in das bestehende HiL-System die Radarsensoren integriert werden. Um eine gemeinsame Umgebungsbasis mit der Stereo-Kamera zu besitzen, ist es zwingend notwendig, die Radarsensoren mit Daten aus der Simulation des Animationsrechners zu speisen. Weiterhin muss eine Design-for-Test (DfT) Schnittstelle definiert werden, die diese Daten dem Radarsensor bereitstellt.

4 Einspeisungsschnittstelle

Zur Spezifizierung einer DFT-Schnittstelle für den Radarsensor werden die wesentlichen Verarbeitungsschritte eines Radarsensors in Abbildung 3 skizziert.

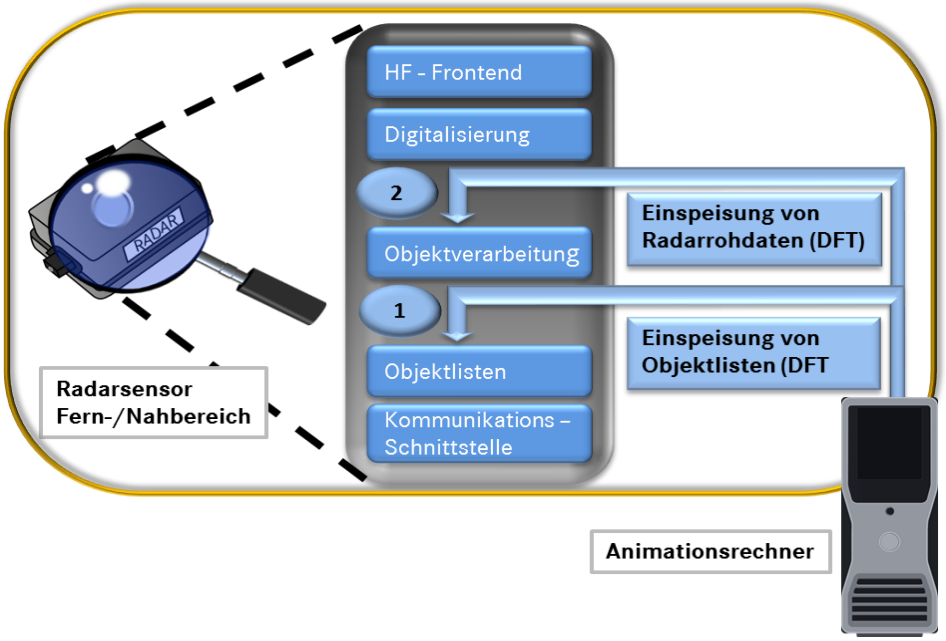


Abbildung 3: DFT-Schnittstellen

Theoretisch kann vor jedem Verarbeitungsschritt eine Einspeisung erfolgen. Wie in Abbildung 3 zu sehen ist, wurden zwei sinnvolle Möglichkeiten für die Einspeisung identifiziert. Diese befinden sich in Punkt 1, nach dem Verarbeitungsschritt Objektverarbeitung und in Punkt 2, nach der Digitalisierung der Radar-Echos.

Bei der Einspeisung von Objektlisten kann nur die Funktionalität der Kommunikationsschnittstelle des Radarsensors getestet werden. Um die Testaussage zu erhöhen und um möglichst viel Funktionalität testen zu können, ist das Ziel, die Daten nach der Digitalisierung einzuspeisen. Deshalb wurde die Design-for-Test (DFT) Schnittstelle für den Einspeisungspunkt 2 definiert.

Durch nähere Erläuterung der einzelnen Verarbeitungsschritte wird deutlich, in welcher Art die Daten nach der Digitalisierung vorliegen. Im HF-Frontend werden elektromagnetische Wellen mit Lichtgeschwindigkeit ausgesendet und während einer Sendepause empfangen. Diese empfangenen Wellen werden von der Digitalisierung abgetastet und die Informationen als Frequenzen abgespeichert. Dementsprechend müssen diese Frequenzen vom Animationsrechner bereitgestellt werden.

Zur Erzeugung dieser Frequenzen zeigt Abbildung 4 eine ausführlichere Systemarchitektur des HiL-Testsystems. Hier werden die Schnittstellen zwischen den drei beteiligten Komponenten HiL, Animations-PC und HiL-Box veranschaulicht. Der HiL simuliert, wie bereits im vorangegangen Kapitel beschrieben, das Ego-Fahrzeug und kommuniziert dessen Position über ein Netzwerkprotokoll an den Animationsrechner(2). Weiterhin werden über dieses Netzwerkprotokoll zusätzliche Steuer- und Simulationsdaten ausgetauscht. Basierend auf den empfangenden Positionsdaten

aktualisiert der Animationsrechner die Umgebung und stellt die aktualisierten Daten den am System beteiligten umgebungserfassenden Sensoren zur Verfügung.

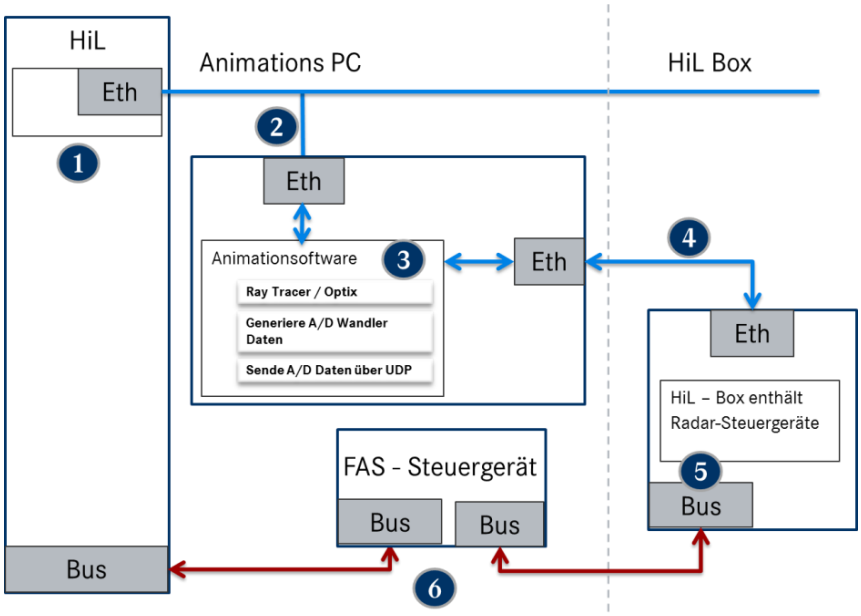


Abbildung 4: Detaillierte Systemarchitektur des HiL-Testsystems

Während der Aktualisierung der Umgebung arbeitet auch der Raytracer (3), welcher die Strahlen simuliert, die vom Radarsensor ausgesendet werden würden. Die erzeugten digitalen Daten müssen dann durch einen Frequenzgenerator in Frequenzdaten umgewandelt werden. Diese Frequenzdaten werden über UDP an die HiL Box (4) geschickt. In der HiL-Box sind die realen Radarsensoren mit einer DFT-Schnittstelle versehen über welche die empfangen Frequenzdaten als Radar-Detektionen in den A/D Wandler Speicher geschrieben werden. Dann beginnt das Radar-Steuergerät auf den eingespeisten Daten seine Erkennungs- und Klassifikationsalgorithmen auszuführen und stellt abschließend eine Objekteliste auf dem Fahrzeugbus bereit. Diese Objektliste wird vom FAS-Steuergerät entgegengenommen und weiterverarbeitet.

5 Erzeugung und Interpolation virtueller Radar Detektionen

Zur Simulation von Radar Detektionen in einer Echtzeit 3D Animation wird das Raytracing-Verfahren eingesetzt. Beim Raytracing-Verfahren (Abbildung 5) wird für jeden Pixel des Ausgangsbildes ein Primärstrahl in die 3D Szene geschossen. Diese Primärstrahlen treffen analog zu den Radarstrahlen auf Objekte in der Szene. Wie in Abbildung 5 veranschaulicht, trifft ein Primärstrahl z.B. auf eine halbtransparente (halbtransparent bezogen auf die elektromagnetischen Materialeigenschaften) Kugel. Basierend auf den Materialeigenschaften, dem Einfallswinkel, der Geometrie des Objekts und der Oberflächenbeschaffenheit berechnet der Raytracer Sekundärstrahlen in

Form von Transmissions- und Reflexionsstrahlen. Diese Sekundärstrahlen werden ebenfalls in die Szene geschossen und können wieder auf weitere Objekte treffen. Dieser Vorgang kann rekursiv wiederholt werden bis der Strahl die Szene verlässt oder ein definiertes Abbruchkriterium wie z.B. die Anzahl an Iterationen erreicht wurde. Das Ausgangsbild (Umgebungs-Scan) hat die Dimension des Sichtbereiches des Radarsensors und enthält die Auftreffpunkte (Detektionen) der Strahlen in Richtung des Radarsensors.

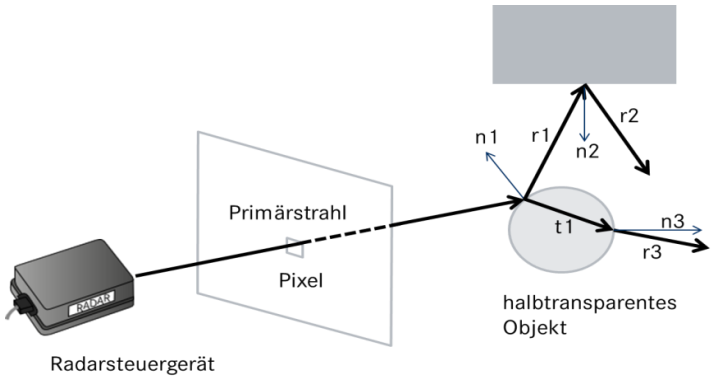


Abbildung 5: Funktionsweise Raytracing

Die benötigten Parameter wie Reflexions- und Transmissionsfaktor werden für jede Textur aus einer XML-Datenbank geladen. Weiterhin enthält die XML-Datenbank Radarquerschnitte für die in der 3D-Animation verwendeten Objekte wie Autos, Häuser, Straßenbelag, Türen, usw… Diese Parameter basieren auf gemessen Daten mit dem realen Radarsensor. Mithilfe dieser Parameter kann die Intensität der Sekundärstrahlen beeinflusst werden, die direkte Auswirkungen auf den Radarquerschnitt der Detektion hat. Sinkt die Intensität eines Strahles unter einen bestimmten Schwellwert, kann die Nachverfolgung auf Grund der geringen Auswirkungen auf das Ausgangsbild abgebrochen werden. Das erhaltene Ausgangsbild ist für den verwendeten Sensor viel zu exakt und muss deshalb durch real gemessene Rauschwerte modifiziert werden.

Obwohl es gelungen ist, das Raytraceing zur Simulation der Radarstrahlen in Echtzeit (60 Hz) zu berechnen, reicht diese Geschwindigkeit zur vollständigen Simulation der Radarstrahlen nicht aus. Zum Vergleich berechnet der Radarsensor ein Umgebungs-Scan ähnlich dem Raytracer in weniger als 1 ms. Weiterhin führt er mehrere Umgebungs-Scans hintereinander aus, sodass er bereits 16-mal die Umgebung erfasst hat, bis der Raytracer, den ersten Umgebungsscan geliefert hat. Hinzu kommt, dass der Radarsensor mit zwei unterschiedlichen Modi arbeitet, die in Bezug auf Reichweite, Öffnungswinkel und Abtastungsraster differieren. Zusätzlich werden noch mehrere Antennen eingesetzt zum Aussenden und Empfangen der Radarstrahlen.

Um dennoch den Hardware Radarsensor am HiL zu testen, muss ein Interpolation-Algorithmus eingesetzt werden, der die fehlenden Umgebungs-Scan interpoliert, die verschiedenen Modi berechnet und die Detektionen unter Berücksichtigung des Doppler Effekts auf die einzelnen Empfangsantennen aufteilt.

Der Raytracer kann dementsprechend parametriert werden, dass er ein Ausgangsbild für beide Modi liefert oder separat für jeden Modus. Anschließend werden die Daten konvertiert um performant über das Netzwerk übertragen zu werden und um die Einspeisung beim Empfang durch die HiL-Box zu erleichtern. Die HiL-Box speist die empfangen Detektionen direkt in den A/D Wandler Speicher ein.

Als Raytracer wird das bereits im Abschnitt „Verwandte Arbeiten“ erwähnte echtzeitfähige Raytracing Framework Optix verwendet.

6 Zeitliche Evaluation für die gesamte Integrationskette

Laut Definition läuft ein HiL-Testsystem immer in Echtzeit. Daraus resultiert, dass die Integrationslösung des Radarsensors diese Anforderung erfüllen muss. Kann die Echtzeit nicht eingehalten werden, hätte das erhebliche Auswirkungen auf die Fusion der Daten. Durch die Verletzung der Echtzeit, wird das Objekt nicht rechtzeitig auf dem Fahrzeugbus bereitgestellt und deshalb ist eine Fusion der Daten nicht mehr möglich. Zusätzlich liest das Radarsteuergerät Daten auf dem Fahrzeugbus mit und überprüft diese durch interne Plausibilitätskontrollen mit den erfassten Daten. Diese Kontrollen würden ebenfalls fehlschlagen.

Aufgrund dessen muss eine exakte Untersuchung des Zeitverhaltens von der Aktualisierung der Umgebung bis hin zur Bereitstellung der Daten auf dem Fahrzeugbus durchgeführt werden.

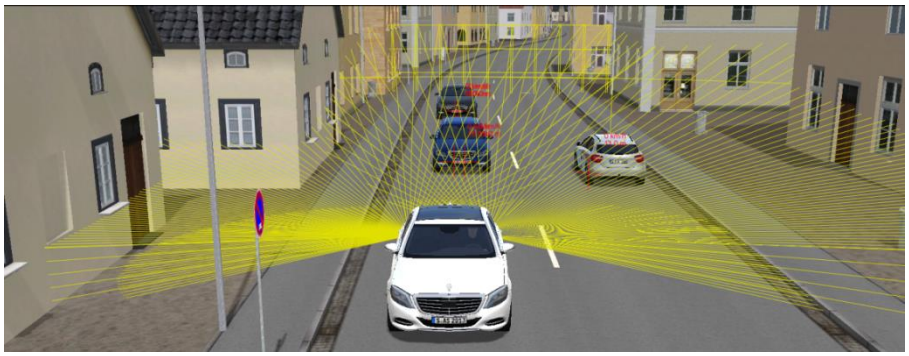


Abbildung 6: Veranschaulichung des Test-Szenarios

Für die Untersuchung wird ein einfaches Test-Szenario mit drei Fahrzeugen verwendet. Das vordere weiße Fahrzeug in der Abbildung 6 repräsentiert das Testfahrzeug (Ego Fahrzeug), dass die zu testenden Sensoren enthält. Bei diesem Szenario führt, das Ego Fahrzeug eine Notbremsung durch, aufgrund eines Fußgängers der in die Straße läuft. Im Diagramm auf der linken Seite der Abbildung 7 sind die Positionen der beteiligten Fahrzeuge veranschaulicht. Der Punkt (0,0) zeigt die Position des Ego-Fahrzeuges. Ausgehend vom Ego-Fahrzeug sind in der gleichen Fahrspur noch zwei weitere Fahrzeuge positioniert. Weiterhin ist ein entgegenkommendes Fahrzeug (-15,-1,2) definiert.

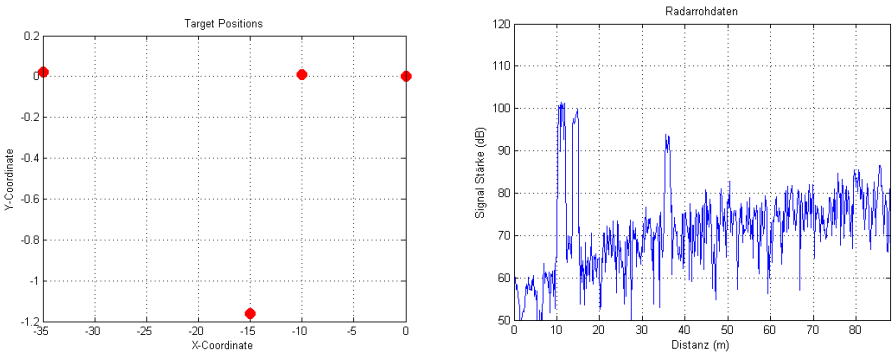


Abbildung 7: Objekt Positionen und Generierte Radar A/D Wandler Daten

Dieses Szenario wird am HiL-Testsystem geladen und ausgeführt. Mittels des Raytracing Verfahrens werden die Radar-Detektionen ermittelt. Diese Detektionen werden über den Interpolations-Algorithmus in Frequenzdaten für den A/D Wandler Speicher konvertiert. Die rechte Seite in Abbildung 7 veranschaulicht die konvertierten Daten. Zu sehen sind die drei Objekte als Peak zu den Distanzen 10, 15 und 35 Meter. Diese Daten werden über eine Netzwerkschnittstelle an die HiL-Box geschickt und direkt in den A/D Wandler Speicher geschrieben.

Angesichts des enormen Zeitverlusts des Interpolations-Algorithmus wurde dieser für mehrere Detektionen untersucht um festzustellen, wie der Algorithmus skaliert. Abbildung 8 dokumentiert die zeitliche Untersuchung für den sequentiellen und parallelen Ansatz. Hier ist zu sehen, dass sich die Anzahl der Detektionen zur benötigten Zeit linear verhalten. Aufgrund der hohen Komplexität des Interpolations-Algorithmus, stellt die Parallelisierung eine große Herausforderung dar. Dabei besteht die Komplexität aus der Verteilung der Detektionen über mehrere Antennen unter Berücksichtigung mehrerer Umgebungsscans und aus der Verarbeitung der Detektionsparameter wie Position, Geschwindigkeit und Radarquerschnitt. Nach einer intensiven Untersuchung auf Parallelisierung gelang es, einzelne Verarbeitungsschritte herauszulösen und in Threads auszulagern. Durch die parallele Ausführung der Thread konnte die Performance deutlich gesteigert werden. Die Auffälligkeit, dass die maximale Performance der Parallelisierung erst ab Detektion 24 erreicht wird, ist auf die Implementierung zur Aufteilung auf die einzelnen Antennen zurückzuführen. Die Auslegung des Algorithmus war für eine große Anzahl an Detektionen geplant. Trotz des positiven Ergebnisses erfüllt der Interpolations-Algorithmus das vorgegebene Kriterium nicht und ist für eine echtzeitfähige Umsetzung nicht geeignet. Das Kriterium ist, die Ausführungszeit des Interpolations-Algorithmus für eine definierte Anzahl an Detektionen unter der Zykluszeit des Radarsensors zu halten. Die Zykluszeit des Radarsensors umfasst die Umgebungsscans, die Signalaufbereitung, die Objekterkennung und -klassifikation sowie die Ausgabe der Objektliste auf dem Fahrzeugbus. Sie ist definiert als ein konkretes Zeitintervall, in dem diese Aufgaben bearbeitet werden müssen.

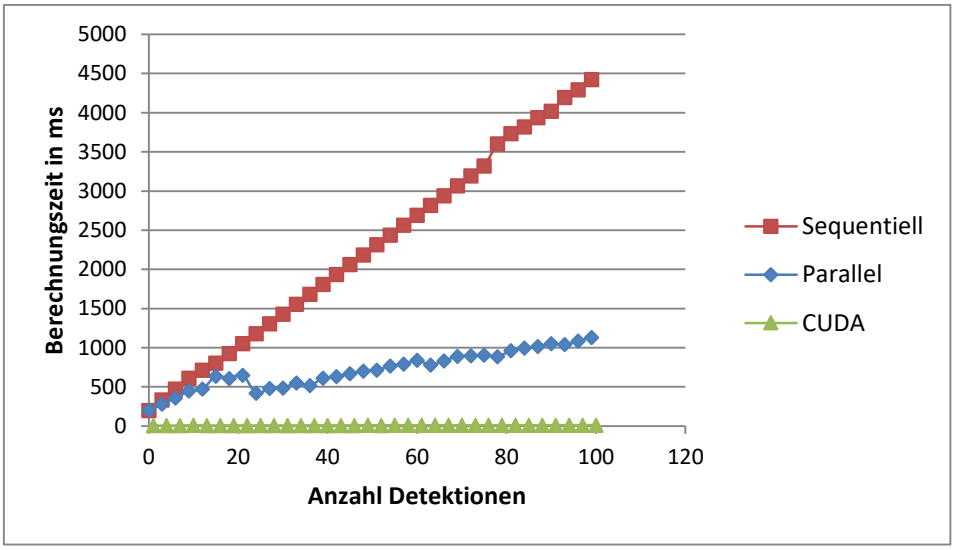


Abbildung 8: Berechnungszeit des Interpolations-Algorithmus

Aufgrund der Verletzung der Zykluszeit musste der Interpolations-Algorithmus weiter parallelisiert, separiert und portiert werden, damit er mittels des Frameworks CUDA auf der Grafikkarte hoch parallelisiert zur Ausführung gebracht werden konnte. Dies gelang mit der Grafikkarte „GFX 480“ für einige Detektionen. Durch die Verwendung einer leistungsfähigeren Grafikkarte „K6000“ konnte die Anzahl der Detektionen gesteigert und der Toleranzbereich für die Ausführung des Interpolations-Algorithmus eingehalten werden(siehe Abbildung 9). Damit ist es möglich, die in Abbildung 4 skizzierte Einspeisungskette innerhalb der Zykluszeit des Radarsensors auszuführen. Dies bedeutete, dass die Berechnung des Raytracers mit Reflexions- und Transmissionsstrahlen, die Konvertierung und Interpolation durch den Interpolations-Algorithmus, die Übertragung zur HiL-Box sowie die Einspeisung in den A/D Wandler Speicher innerhalb eines Zyklus erfolgt. Durch die Einhaltung der Zykluszeit ist der Interpolations-Algorithmus für eine echtzeitfähige Umsetzung geeignet.

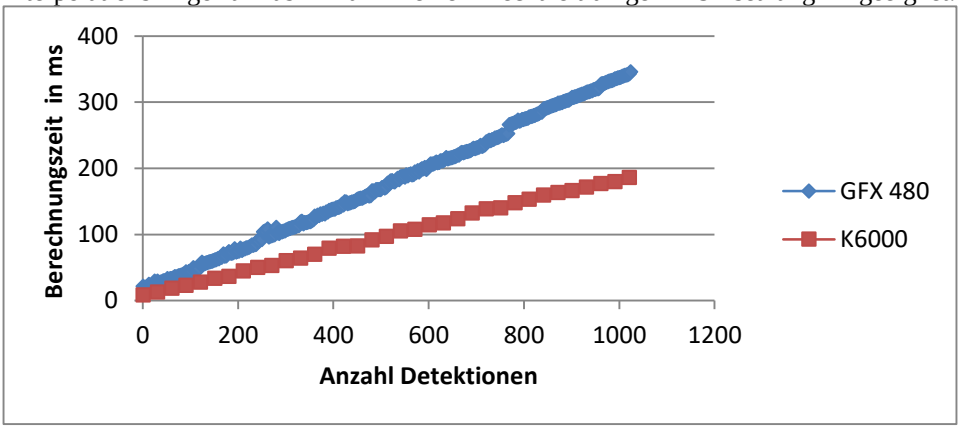


Abbildung 9: Berechnungszeit des CUDA optimierten Interpolations-Algorithmus

7 Validierung der Gesamtkette

Im vorangegangenen Kapitel wurde gezeigt, dass es aus zeitlicher Sicht möglich ist, virtuelle Radardetektionen zu generieren und rechtzeitig in den realen Radarsensor einzuspeisen. Nach der Einspeisung in den Sensor führt dieser Erkennungs- und Klassifikationsalgorithmen aus und liefert abschließend auf dem Fahrzeugbus eine Objekteliste mit relevanten Objekten. Im Zug der Validierung wurde das für die Zeitmessung verwendete Testszenario in der realen Welt nachgestellt. Zu diesem realen Testszenario wurden die Objektlisten aufgezeichnet, welche der Radarsensor über den Fahrzeugbus sendet. Diese aufgezeichneten Objektlisten werden mit den Objektlisten basierend auf den virtuell eingespeisten Detektionen verglichen. Dadurch kann validiert werden, ob diese Listen identisch sind. Stellt sich heraus, dass die Listen nach wiederholtem Vergleichen immer noch kongruent sind, kann für dieses Szenario behauptet werden, dass es im Labor ausreichend abgesichert werden kann. Um eine generelle Aussage zur ausreichenden Absicherung im Labor treffen zu können, müssen noch weitere verschiedene Testszenarien mit den identischen Testszenarien im Labor verglichen werden. Eine konkrete Validierung folgt.

8 Fazit

In diesem Beitrag wurde ein neuwertiger Lösungsansatz zur Einspeisung von virtuellen Radardetektionen in einen realen Radarsensor in Echtzeit unter den gegebenen Rahmenbedingungen präsentiert. Somit können Testszenarien mit höchsten 5 beteiligten Objekten im Labor unter HiL Bedingungen getestet werden. Der eingesetzte Interpolations-Algorithmus lässt sich auf mehrere Grafikkarten aufteilen. Dadurch ergibt sich für zukünftige Untersuchungen die Möglichkeit, diesen Aspekt zu untersuchen um eine weitere Performancesteigerung zu erreichen und so das Limit für 5 Objekte zu erhöhen.

Literaturverzeichnis

[Ba96] Bair, George L.: Airbone radar simulation. Camber Corporation 1996

[BB06] Bühren, Markus; Bin, Yang: Simulation of Automotive Radar Target Lists using a Novel Approach of Object Representation. Intelligent Vehicles Symposium 2006. S. 314-319

[BB07] Bühren, Markus; Bin, Yang: Extension of Automotive Radar Target List Simulation to consider further Physical Aspects. Telecommunications 2007; S. 1-6

[Be13] Belbachir, Assia et. al.: Simulation-Driven Validation of Advanced Driving-Assistance Systems. Procedia-Social and Behavioral Sciences, 2012, 48. Jg., S. 1205-1214.

[Ka86] Kajiya, James T.: The rendering equation. SIGGRAPH 1986.

[NSM11] Nentwig, Mirko; Schieber, Reinhard; Miegler, Maximilian: Hardware-in-the-Loop-Test für vernetzte Fahrerassistenzsysteme. ATZ elektronik, 04/2011, 6. Jahrgang, S. 20-25

[Pa10] Parker, Steven et. al.: Optix: a general purpose ray tracing engine. ACM Trans. Graph. 2010.

[PLK08] Peinecke, N.; Lueken, T.; Korn, B. R.: Lidar simulation using graphics hardware acceleration. Digital Avionics Systems Conference 2008

[Sh12] Shuiying, Wang et. al.:. Shader-based sensor simulation for autonomous car testing. Intelligent Transportation System 2012. S. 224-229

[Su11] Sume, A et. al.: Radar Detection of Moving Targets Behind Corners, Transactions on Geoscience and Remote 2011.

[WWS10] Wohlfahrt, Christoph; Weizenegger, Florian; Smuda, Peer: HiL-Testtechnologie für kamerabasierte Fahrerassistenzsysteme. AutoTest 2010

Extending Software Architectures from Safety to Security

Martin Böhner, Alexander Mattausch, Alexander Much
CIS-Technology
Elektrobit Automotive GmbH
Am Wolfsmantel 46
91058 Erlangen
martin.boehner@elektrobit.com
alexander.mattausch@elektrobit.com
alexander.much@elektrobit.com

Abstract: In this paper we summarize approaches for software architectures used in the automotive domain for safety-critical or mixed safety-critical systems and extend the approach to security-critical systems. Safety and security aspects of systems influence each other and we show solutions which combine both worlds in a common architectural and development process approach.

1 Motivation and Background

In recent years several studies (e.g. [KCR+10], [MV13], [Dan10]) revealed a growing security threat potential for cars from all kinds of aspects, e.g.

- System and software complexity: e.g. functionality per ECU, lines of code
- Connectivity: e.g. WiFi, Bluetooth, Ethernet and Internet, Car2Car, Car2X
- Interoperability: e.g. WiFi, Bluetooth, Internet, mobile phone integration
- Sped up time-to-market due to the necessity to adapt quickly: e.g. Apps, 3rd party SW

For future applications in the automotive domain the topics connectivity and collaboration with various outside sources are essential features. On the one hand this includes communication with various sensors and applications in the car itself. On the other hand, however, this will also comprise external sources such as mobile devices, roadside infrastructure or backend IT systems that provide cloud computing capacities.

The signals exchanged here will be utilized in safety and security critical use cases like assisted and autonomous driving, or in the exchange of traffic and hazard information between cars and roadside infrastructure.

As shown by different independent studies the mutual influence between safety and security aspects is very strong. Safety software engineering and the resulting applications

Figure 1: Connected car with various communication partners

have a direct impact on security and vice versa. This has been recognized by established software safety standards like IEC 61508 [IEC10] or ISO 26262 [ISO11b]. And it has also been addressed by regulators of other domains, e.g. by the FDA jointly with the ICS-CERT in 2013: *"Manufacturers are responsible for remaining vigilant about identifying risks and hazards associated with their medical devices, including risks related to cybersecurity, and are responsible for putting appropriate mitigations in place to address patient safety and assure proper device performance"* [FDA13], [IC13]. Even regulatory bodies or political institutions like the European Commission correlate both topics, e.g. José Manuel Durão Barroso in a speech about nuclear safety in 2012: *"[. . .] there is no safety without security and vice-versa"*, see [Jos12].

There are already initiatives by standardization bodies to identify synergies in creating the assurance cases that combine approaches like safety cases, security assurance and system integrity levels, e.g. ISO 15026 [ISO11a].

A practical consequence for today's systems is that most safety critical use cases will have to consider intentional manipulation of messages and data as well as direct attacks on software and hardware level. Such systems need to be accompanied with a state of the art argumentation for safety and security, see e.g. the BGH airbag case [Bun09]. However, the practical interpretation of "state of the art" is changing quickly and system development needs to take this background into account.

This article presents practical approaches for software architectures addressing the integrity of automotive ECUs using established environments like AUTOSAR (see [AUT14a]). Such software architectures provide methods to address safety, security and availability aspects of today's state of the art ECUs.

2 Integrity and Safety Mechanisms

The basic software architecture of a safety-critical system is responsible for providing mechanisms to establish the integrity of the system from a software point of view. Such mechanisms also include the critical interfaces between hardware and software, including detection mechanisms of hardware faults. AUTOSAR provides a framework that allows Software Components (SW-Cs), as the building blocks of a full AUTOSAR application are called, to concentrate on safety-critical functions independent of the practical implementation of the basic software layers.

The provision of such basic mechanisms is well established, e.g. in the definitions of "independence" in IEC 61508 [IEC10] or *freedom from interference* or *criteria for coexistence of elements* in ISO 26262 [ISO11b]. Technically the "best" definition can be found in DO-178C (see [RTC11], 2.4). It defines three classical architectural considerations: partitioning, multi-version dissimilar software and safety monitoring.

This chapter mentions these methods briefly as reference. The focus of this paper is on the extension of these safety mechanisms to security mechanisms. Further details can be found in [MA09], [HJMA14], [GM18].

2.1 Memory Partitioning

In most software architectures memory partitioning is provided by the operating system. For safety-critical systems this includes at least memory write protection to separate different tasks from each other. State of the art operating systems also strictly separate the operating system kernel from all other parts of the software system. This usually results in a software architecture with a clear separation of safety monitors from the monitored safety functions as well as a separation of multi-version dissimilar software.

State of the art operating systems also provide stack protection features to prevent different tasks from writing to stacks of other tasks and to prevent or at least limit stack overruns. Additionally, the execution context of each task is encapsulated in the operating system to prevent write access from other tasks.

Memory partitioning is based on hardware features such as memory protection units (MPU) or memory management units (MMU). The level of protection provided by the operating system varies with the different hardware implementations.

See Figure 2 for a sample software architecture including protection mechanism for safety-critical ECUs using the AUTOSAR software architecture as basis.

2.2 Execution and Scheduling Protection

Independence in software has two aspects: spatial and temporal independence. Memory protection and a good analysis of the hardware-software interactions including reactions

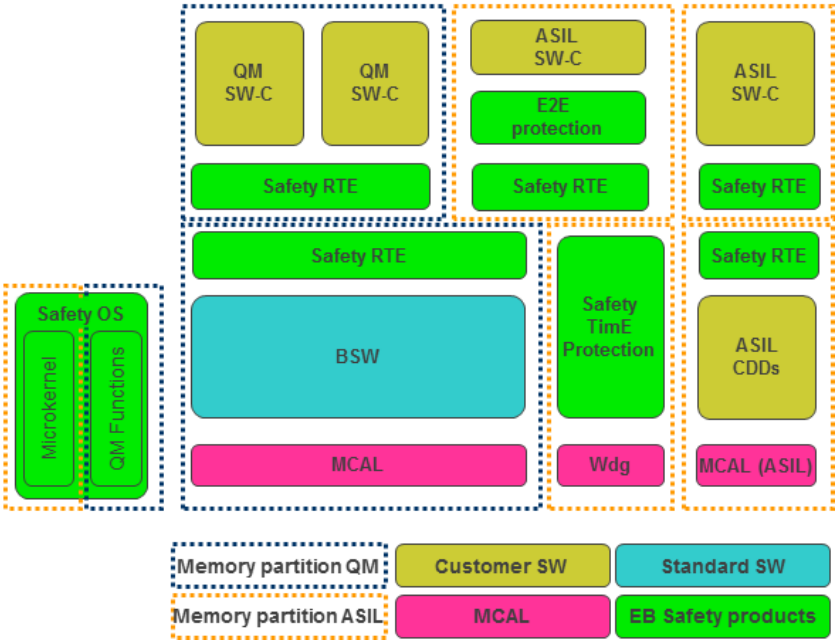


Figure 2: Safety architecture for an AUTOSAR ECU

to faults can provide spatial independence.

Temporal independence is based on mechanisms such as time monitoring, e.g. budget and deadline monitoring, or execution monitoring, e.g. alive supervision and control flow monitoring. These safety mechanisms are implemented in the operating systems or as a safety mechanism which is also independent and separated from the operating system.

2.3 Message Protection

Protection of messages is defined also in chapter “D.2.4 Exchange of Information” of ISO 26262 [ISO11b]. Failure scenarios for safety-critical ECUs include corruption, repetition, loss, delay of information.

The most commonly implemented approach is an end-to-end protection of the messages and data transmitted within an ECU or between ECUs. In an AUTOSAR context this enables the application to stay independent of the communication mechanisms and communication busses.

An implementation of the AUTOSAR end-to-end protection library including some extensions, which is developed according to SIL-3 and ASIL-D, provides the commonly used safety mechanisms for message protection.

3 Security Mechanisms

Security-critical systems have different needs than safety-critical systems as the "opponent" changes. The intention of a safety-critical system is to prevent harm emanating from the system itself whereas a security-critical systems intent to prevent harm to the system from outside, e.g. from a malicious human attacker.

In ideal world the "threat model" of a safety-critical system can be determined up-front during system design, analysis and implementation. The risk emanating from a safety-critical system is usually based on systematic or random hardware faults or systematic faults in software. Such a threat model is iteratively enhanced to include previously unknown faults, but basically one could call it "static".

Security-critical systems however need to deal with "dynamic" threat models as the methods used by an attacker are not known during system creation and can significantly improve during system operation. The major factors are long-term use of automotive systems, evolution in computing power, but most importantly the ingenuity of the attackers. Usually this makes security-critical systems more complicated to build and analyze than safety-critical systems.

However, the responsibility of the basic software doesn't change: provide mechanisms for ensuring the integrity of the system. This can be achieved by extending software architectural mechanisms established for safety-critical systems as well as adding new mechanisms especially designed for security topics. Most importantly, synergies can arise from the two analyses, as both analysis methods are risk-based. The detection of security breaches can also uncover possible safety issues and vice-versa. An example are typical stack overflow attacks, as discussed in more detail below. This chapter extends basic software architectures designed for safety-critical systems to security-critical systems.

3.1 Memory Read and Execution Protection

A good example for combined safety and security threats are stack overflows. They either originate from a fault in the software, which would be a safety issue, or they can be provoked by an external attacker. In both cases, such a scenario is part of the analysis and needs to be prevented by design or at least be detected during run-time.

Our architecture proposal shows solutions for many common safety and security topics. Staying with the stack overflow example, the EB tresos Safety OS prevents stack overflows using memory protection. While this still gives rise to possible denial-of-service attacks, it is not possible to bring the system into an undefined state. The overflow is detected and prevented, and the microkernel of the Safety OS invokes a pre-defined error handling ensuring that the ECU is transferred into a well-defined safe state or a degraded mode. The freedom-from-interference argument that is commonly used in safety-related mixed-criticality systems can thereby be used to ensure the separation of the critical safety mechanism from the vulnerable software part that is exposed to the "outside world".

Besides the MPU's write protection mechanisms, which is the basic use case for safety-critical software, modern CPUs also allow to activate read protection and even execution protection. The former can be used to ensure that security-critical data is only accessible by those software components that actually need it. This data can e.g. contain encryption keys or authentication data that must be protected against the software parts that have access to the network. A recent example for such an exploit is the "Heartbleed" vulnerability of OpenSSL, where the attacker can read sensitive authentication data via a maintenance function that is entirely unrelated to OpenSSL's encryption functionality.

The execution protection provides a further barrier against the attacker: the code dealing with the security-related data can be marked as non-executable, unless it is actually scheduled to run. In addition, adding execution protection to the stack also closes a commonly used door for intrusions into systems. Both mechanisms together thus eliminate two possible attack vectors: First, making unrelated code read and expose data that it is not considered to be read. Second, trying to execute the code involuntarily from a different context is actively prohibited.

3.2 Message Authentication

As already mentioned in chapter 2.3 in the safety context, a major issue is the reliability of message data. Since the scope of "opponents" is broadened in the security context, one has to assume an attacker has control over all communication channels in the sense of an attacker model according to Dolev-Yao (see [DY29]). This assumes particularly that an attacker can:

- Obtain any message passing through the network and read, modify or delete it
- Impersonate a legitimate user and thus is able to initiate a communication with any other user
- And therefore replay or delay messages

In the following we use the definitions for data integrity, authentication, entity authentication, data origin and message and transaction authentication given in [MVO96].

3.2.1 Authenticity and Integrity

Derived from the attacker model and according to [AUT14b], chapter 3.2 and 4.2 the major goals from a security perspective for message protection can be summarized as authenticity of entities and messages, data integrity, data freshness and message confidentiality (see chapter 3.3).

Data origin authentication or message authentication techniques provide assurance to a receiving party of the identity of the party which originated the message. Data origin authentication implicitly provides data integrity, since, if the message was modified during

transmission, communication partner ‘A’ would no longer be the originator (see [MVO96], chapter 1.7).

The most commonly used technique which is also proposed as the default method in [AUT14b] to achieve message authentication and integrity is a Message Authentication Code (MAC) and more specific a CMAC [Dwo05] based on AES [UDoCN01] with an adequate key length. As shown in Figure 3 the message on the bus is concatenated with (a part) of the MAC on sender side and can be verified on the receiver side.

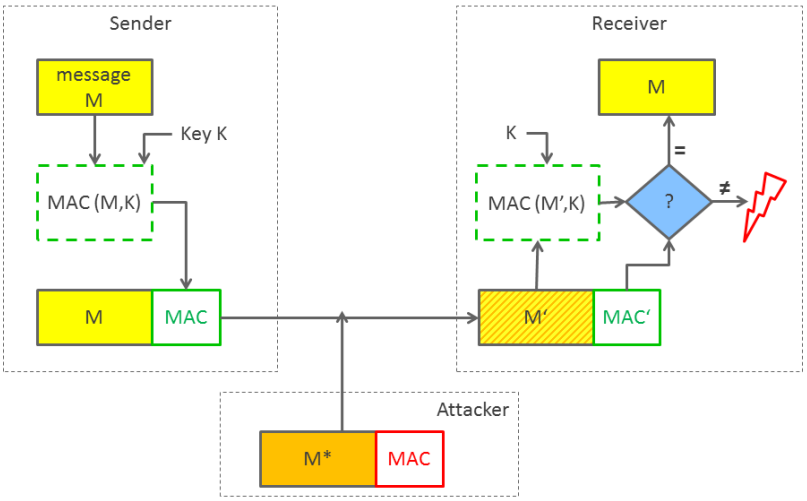


Figure 3: Message authentication code

3.2.2 Freshness

Using MACs alone does not protect from messages being recorded at a certain point in time and played (delay) or replayed by an attacker later. To prevent that an attacker simply records an authentic message of integrity and replays it at any point in time, each proof of authenticity has to be unique. This is achieved by adding a so called freshness value into the MAC generation process on sender side which allows the receiver to detect replayed messages on his side. Different sources can be used as freshness value, e.g. a monotonic counter or a reliable timestamp value (see [AUT14b]).

An implementation of the AUTOSAR concept ‘Secure Onboard Communication’ (see [AUT14b]) provides the commonly used security mechanisms for message protection ensuring authenticity, integrity and freshness of messages.

3.3 Message Privacy

At the moment, the AUTOSAR Secure Onboard Communication concept does not contain default use cases requiring message privacy. However increasing connectivity of vehicles and at the same time increasing integration of personal devices like smartphones into the vehicle environment will lead to a growing need for solutions providing privacy for various data.

Message privacy can be obtained by using, either symmetric or asymmetric (public key cryptography) encryption methods.
The most well-known and thoroughly analyzed symmetrical encryption method for current use cases is the Advanced Encryption Standard (AES) [UDoCN01]. State-of-the-art solutions for ECUs with higher security requirements use cryptographic functionality implemented in hardware, e.g. Secure Hardware Module (SHE) or more general Hardware Security Modules (HSM). Due to the standardized layered architecture such hardware peripherals can easily be made available in an AUTOSAR stack.

For asymmetrical cryptography, there are standards like RSA and elliptic curve cryptography [MVO96]. Compared to symmetric algorithms they are more computationally intensive and no standardized cryptographic hardware peripheral for automotive domain is currently available.

4 Extended Safety and Security Architecture

As seen in chapters 2 and 3 message protection plays an essential role in safety as well as security related applications. For both aspects protection against corruption, repetition, loss and delay of information must be ensured.

However the goals of both protection methods differ. For safety aspects the path from the originating to the receiving software application shall be protected against transmission and processing errors both on bus level as well and particularly as throughout the basic software stack on sender and receiver side. Message protection mechanisms with regard to security aim to protect messages on the bus against manipulation attacks. Protection within the software stack itself is not a goal of currently proposed security mechanisms (see [AUT14b]).

Therefore different methods are used to address safety and security aspects independently from each other. For protection against safety issues a checksum based end-to-end (E2E) protection mechanism specified by AUTOSAR (see [AUT13]) is most commonly used.

Protection against malicious attacks is done as specified in [AUT14b] and explained in sections 3.2.2 and 3.2.1 by appending and transmitting (parts of) a MAC value to the actual message data.

If a message is both safety and security relevant at the same time, currently two individual and independent attributes of significant length have to be added to the actual message data.

4.1 Safety and Security Architecture for Message Protection

To optimize the usage of available and already scarce bandwidth we propose the following two architectures for combined message protection with regard to safety and security. Since a MAC incorporating a freshness value can cover all aspects provided by a CRC value and extend these attributes by security relevant extensions as described in chapter 3 a MAC is used to protect messages on bus level.

The remaining challenge is to close the protection gap for safety protection between the verification of the MAC and the sending/receiving application.

4.1.1 Autosar E2E and SecOC modules

The safety protection gap arises because Autosar Secure Onboard Communication (SecOC) module is located on PduR level. To protect the path from PduR to the receiving application (usually a SWC above the RTE) we use the already established E2E mechanisms as shown in figure 4. On the bus only the security extension using a MAC is present.

Therefore SecOC does not simply cut off the MAC after successful verification - the default behaviour of SecOC – but converts it to a safety CRC. The application can then verify the faultless reception of the message in the same way as if the message is just safety relevant and protected by E2E mechanisms on the complete path.

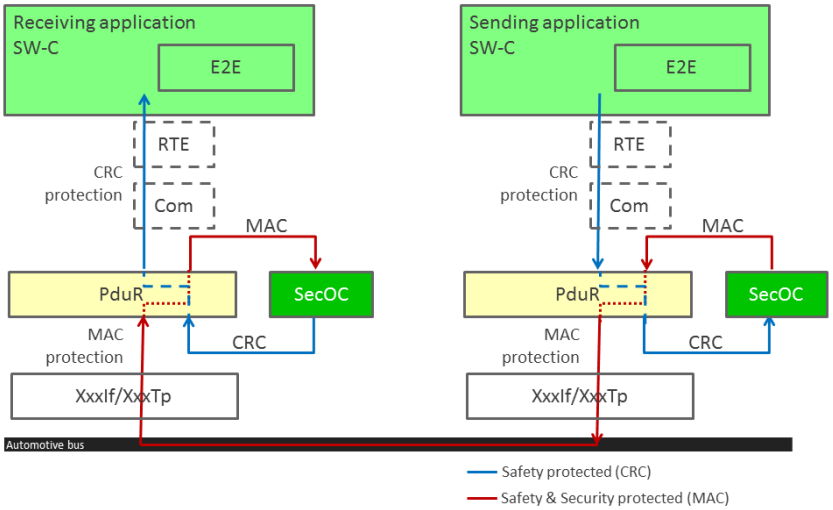


Figure 4: Architecture combining E2E and SecOC modules to efficiently protect messages

This architecture is compatible to both safety and security concepts in AUTOSAR enabling the use of all scenarios and features of the original solutions. It extends the SecOC just to transform a MAC into a CRC value and vice-versa.

4.1.2 End-to-end protection with security methods

An alternative setup EB specialists use to realize end-to-end protection with safety and security allocation is shown in figure 5. Here the safety module is replaced by the security module. To protect a message also through the basic software stack the latter is placed close to the application. AUTOSAR 4.2.1 plans to integrate E2E as part of the serializer/transformer chain ([WA14]). Figure 5 shows both architecturally similar approaches for such a scenario, with the new version for AUTOSAR 4.2.1 on the right-hand side.

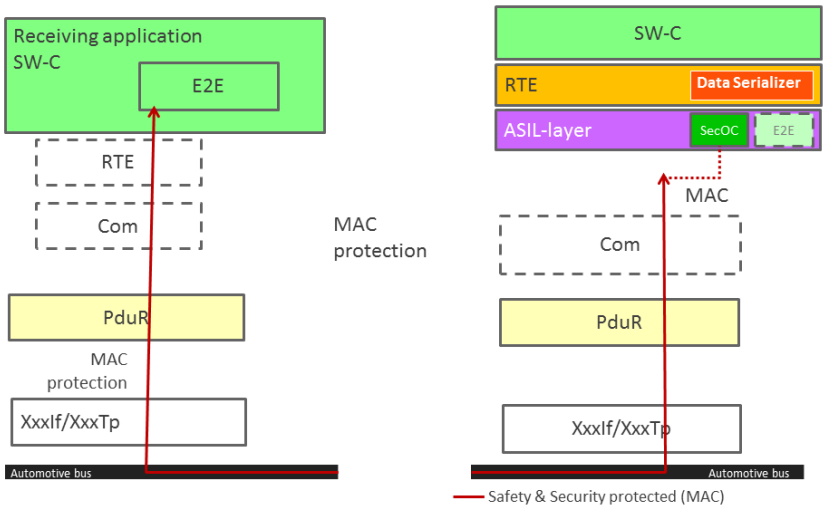


Figure 5: Architecture replacing E2E safety with security mechanisms

This approach, while simpler on paper, requires to change the integration of several adjacent but crucial disciplines like key management, communication behaviour etc. compared to the original standardized concepts.

4.2 Extended Safety and Security Architecture

An architecture combining message protection with the memory read and execution protection mechanisms described in chapter 3.1 is shown in figure 6.

In our implemented application the message protection variant from chapter 4.1.1 was combined with memory read and execution protection provided by EB safety products. To calculate and verify the CMAC we used a SHE hardware peripheral which was integrated into the AUTOSAR stack via a so called CryShe module.

The approach allows to mix security and safety software (also different (A)SIL level) as well as software with normal quality requirements (QM) and enables the integration of

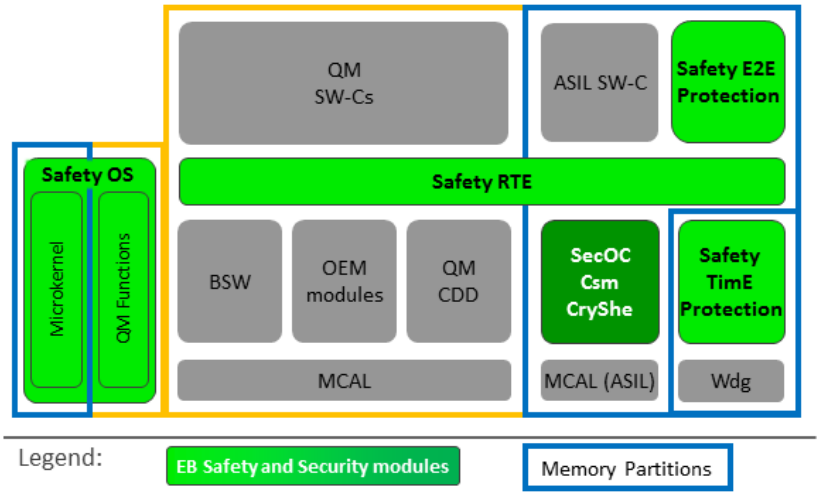


Figure 6: Architecture combining safety with security mechanisms for message, memory and execution protection

"black-box" software (e.g. from an OEM or another supplier) while re-using most building blocks of the standard architecture.

5 Systems and Software Engineering

For safety-critical systems techniques for performing safety analyses are well-established and these are based on approaches known from risk management combined with specific analysis techniques, e.g. for software architectures or the implementation itself. Risk analyses of the system and the implementation are the most important "add-on" in the development process of safety-critical systems compared to standard development processes.

Security analysis of systems is already established in other domains, e.g. in the IT sector. For many automotive ECUs a dedicated security analysis is often not part of the standard development processes yet. However both analysis types are similar in the sense that they are risk-based. These analyses become more alike the closer they are performed to the actual implementation: starting with the software design the formal inspections are nearly identical.

Both domains require rigid software engineering and many threats are already prevented using established processes and methods. For a good summary see a recent response to the heartbleed issue by David Wheeler [Whe14].

When engineering security-critical systems a very important aspect is to look at the complete chain allocating security requirements. The system in question is never only the

embedded target but affects all modules communicating with the functions in question as well as providing and storing crucial data, e.g. key management and key distribution systems, IT infrastructure, cloud services, etc. As a consequence the IT security in the companies needs to be linked with the devices.

A good methodological starting point for risk identification, analysis and risk treatment is the Information Security Management System (ISMS) standard – ISO 27001 ([iso14]).

6 Conclusion

In this paper we presented a practical approach to extend a software architecture designed for safety critical systems based on AUTOSAR to be used in security critical systems.

We have pointed out the mutual influence between safety and security aspects which are recognized already in different standardization groups as well as by regulatory bodies and political institutions. José Manuel Durão Barroso summarized them spot on by pointing out "[...] there is no safety without security and vice-versa", see [Jos12].

Already well-established mechanisms in safety development, e.g. in the definitions of "independence" in IEC 61508 [IEC10] or "freedom from interference" and "criteria for coexistence of elements" according to ISO 26262 can be reused. For applications in common security use cases they are extended by cryptographic methods to ensure e.g. entity and message authentication as well as advanced integrity checks.

We have shown, that one can achieve a significantly increased level of safety as well as security by combining already available solutions from both worlds and consequently applying rigid software engineering processes and methods derived from functional safety development and extending them to security use cases.

7 Outlook

We are currently extending the software architecture approach further with a secure hypervisor to support different operating systems to further support popular "apps" which can be used directly in a vehicle. This also includes merging different domains, e.g. AUTOSAR and Linux-based systems as shown in Figure 7 which can communicate in a controlled (secured) way via an inter OS communication module.

In contrast to a safety architecture the adaptation to new security threats will be a never ending process which needs constant monitoring and adaption to newly emerging threats. This implies the capability of the system to be upgradeable and maintainable during the whole operative life-cycle, which is interesting especially from a safety-critical point of view.

With the development of more and more connected features relying on data from various sources this trend will accelerate further and will increase the need for combined safety

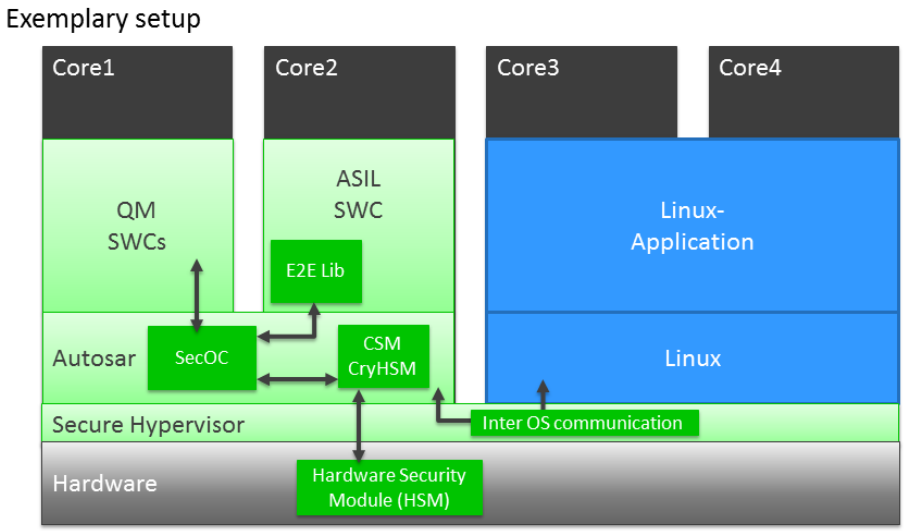


Figure 7: Possible future safety and security architecture supporting different operating systems

and security approaches. These need to be based on standardized software architectures that provide mechanisms for the “integrity” of the system.

Security related systems differ from today’s software architectures used in the automotive domain which are usually based on direct and synchronous API calls. However for memory read protection many API functions need to be converted to asynchronous calls. This change is also motivated by moving from single-core to multi-core processors. The resulting software architectures are challenging since a fundamental property of a safety related system is a limited complexity of the dynamic aspects of the software architecture, e.g. a clearly defined or even static schedule.

References

[AUT13] AUTOSAR. AUTOSAR specification of SW-C End-toEnd Communication Protection Library, 2013.

[AUT14a] AUTOSAR. AUTOSAR - AUTomotive Open System ARchitecture, 2014.

[AUT14b] AUTOSAR. AUTOSAR concept - Secure Onboard Communication, 2014.

[Bun09] Bundesgerichtshof. BGH, Urteil vom 16. Juni 2009, Az.: VI ZR 107/08, 2009.

[Dan10] Christian Daniel. Angreifbare Mobilität: Wie ”online” ist ein modernes Auto und welche Sicherheitsrisiken ergeben sich?, 2010.

[Dwo05] Morris J. Dworkin. SP 800-38B. Recommendation for Block Cipher Modes of Operation: The CMAC Mode for Authentication. Technical report, Gaithersburg, MD, United States, 2005.

[DY29] Danny Dolev and Andrew C. Yao. On the security of public key protocols. 29, 1983-03-29.

[FDA13] FDA. FDA security alert, Cybersecurity for Medical Devices and Hospital Networks, 2013.

[GM18] M. Galla, Thomas and Alexander Much. Targeting Freedom from Runtime Errors for AUTOSAR-Based ECU Software. Mathworks Automotive Conference, 2012-04-18.

[HJMA14] David Haworth, Tobias Jordan, Alexander Mattausch, and Much Alexander. Freedom from Interference for AUTOSAR-based ECUS: a partitioned AUTOSAR stack. Automotive Safety and Security, 2012-11-14.

[IC13] ICS-CERT. ICS-CERT ICS-ALERT-13-164-01 - Medical Devices Hard-Coded Passwords, 2013.

[IEC10] *Functional safety of electrical/electronic/programmable electronic safety-related systems: part 6 : guidelines on the application of IEC 61508-2 and IEC 61508-3.; 2.0*. IEC, Geneva, 2010.

[ISO11a] ISO. ISO 15026, Systems and Software Engineering – Systems and Software Assurance, 2011.

[ISO11b] ISO. Road vehicles – Functional safety, 2011.

[iso14] ISO/IEC 27001:2013/Cor 1:2014: Information Security Management System (ISMS) standard. *Online: http://www.iso.org/iso/iso27001*, 2014.

[Jos12] EU Action on Nuclear Safety, European Commission - SPEECH/12/227 27/03/2012, 2012.

[KCR+10] Karl Koscher, Alexei Czeskis, Franziska Roesner, Shwetak Patel, Tadayoshi Kohno, Stephen Checkoway, Damon Mccoy, Brian Kantor, Danny Anderson, Hovav Shacham, and Stefan Savage. Experimental Security Analysis of a Modern Automobile. Seattle, Washington 98195–2350, 2010. In Proceedings of IEEE Symposium on Security and Privacy.

[MA09] Alexander Mattausch and Much Alexander. Standardized Software Architectures for Safety ECUs. Forum Funktionale Sicherheit, 2013-10-09.

[MV13] Charlie Miller and Chris Valasek. Adventures in Automotive Networks and Control Units. *Last Accessed from h ttp://illmatics. com/car_ hacking. pdf on*, 13, 2013.

[MVO96] Alfred J. Menezes, Scott A. Vanstone, and Paul C. Van Oorschot. *Handbook of Applied Cryptography*. CRC Press, Inc., 1996.

[RTC11] RTCA. DO-178C – Software Considerations in Airborne Systems and Equipment Certification, 2011.

[UDoCN01] National Institute of Standards U.S. Department of Commerce, Information Technology Laboratory (ITL) and Technology (NIST). FIPS Pub 197: Advanced Encryption Standard (AES). Technical report, Gaithersburg, MD, USA, 2001.

[WA14] AUTOSAR WP-A3. AUTOSAR concept - E2E extension, 2014.

[Whe14] David A. Wheeler. How to Prevent the next Heartbleed, 2014.

Safety Issues of Integrating IVI and ADAS functionality via running Linux and AUTOSAR in parallel on a Dual-Core-System[1]

Jörn Schneider, Tillmann Nett

Department of Computer Science
Trier University of Applied Sciences
Schneidershof
54293 Trier
J.Schneider@Hochschule-Trier.de
T.Nett@Hochschule-Trier.de

Abstract: The tight integration of In-Vehicle-Infotainment (IVI) applications and Advanced Driver Assistance Systems (ADAS) or even semi-automated driving functionality on the same hardware offers new chances for innovations in the automotive domain. One of the major challenges in this respect is to achieve a solution that satisfies the heterogeneous requirements from the involved application and operating system worlds and guarantees the necessary freedom from interference to attain the required safety according to ISO 26262 [ISO11]. This paper presents a prototypical solution and discusses the remaining challenges for the chosen virtualization-less approach.

The prototype was developed for the practical use in the research project econnect Germany and successfully used in electric vehicles of a field study in Trier. An open source variant of AUTOSAR OS and Linux run together on the same processor of the system. Each operating system uses it own processor core. The chosen solution allows for the interaction between the different applications of the two operating systems and requires no virtualization layers thus avoiding additional resource demand and communication latencies.

1 Introduction

A huge number of innovative functions in modern vehicles utilize the combination of formerly separated functional domains. Since the introduction of the CAN bus into vehicles ever more communication links have been established in order to enable interaction between previously isolated functionalities. A new area of functional interaction with promising possibilities are In-Vehicle-Infotainment (IVI) and Advanced

[1] This work is partly funded by the German Federal Ministry for Economic Affairs and Energy under grant number 01ME12043 within the econnect Germany project.

Driver Assistance Systems (ADAS) or other safety-related vehicle functions. Apart from this area there are further classes of future systems that could benefit from the integration of open source operating systems such as Linux or Android and AUTOSAR applications on the same hardware. Developing such systems confronts the industry with the challenge of integrating a general purpose operating system and its applications with a real-time operating system and its safety-related applications.

Within the econnect Germany project a prototype of a system running Linux and AUTOSAR on separate cores of the same processor was developed and used for a field study in five modified electric series cars. Although the system's intended functionality was different from an ADAS the basic challenges are quite similar. In this paper we investigate the suitability of the chosen approach for combining IVI and ADAS functionality or similar cases with safety-related real-time applications and general purpose functionality. This is done by summarizing the general challenges in Section 3, describing the chosen realization and identifying remaining issues in Section 4, and discussing the overall concept in comparison to other solution approaches in the final Section. Additionally, the initial usage background of the developed prototype system is given in Section 2 to allow for a better understanding of the chosen design.

2 Usage Background of the Prototype

The current system was implemented during the project econnect Germany for a field study in the City of Trier. The goal of the field study is to evaluate the user acceptance of Grid2Vehicle (G2V) and Vehicle2Grid (V2G) technologies for electric vehicles. Widespread deployment of electric vehicles poses great challenges for the energy sector, e.g., due to high and unbalanced loads on electric power networks. However, electric vehicles and renewable energies provide also great synergies [KT05]. With electric vehicles the fluctuating generation of renewable energies can be buffered. Charging of the vehicles may be adapted to current production. Using V2G technology, it will for instance be possible to provide energy from the vehicle batteries to the grid to compensate for low production.

Using V2G and G2V technologies necessitates a distributed IT architecture with an essential part in the vehicle itself. For instance the planning of the charge processes should be based on current and expected energy production, the conditions in the vehicle (e.g. state of charge) and the settings by the driver [NS14]. The latter requires an on-board computer with an interface through which the driver is able to input parameters such as needed range and time of departure. Furthermore, this on-board computer must plan and execute the charge processes. Both for planning as well as for execution of the charge processes a direct communication with other hardware within the vehicle is needed. For example to execute the charge process the system must communicate with the battery management system. Additional requirements for the on-board computer in the field study originate from the aim of evaluating end user acceptance. For instance, GPS coordinates of all trips were recorded.

Such a system includes parts with safety requirements. Although the functionality of the prototype itself is not safety-relevant, the communication on the main vehicle bus must not be adversely influenced as this might violate safety goals of other systems. Moreover, communication within the vehicle often poses hard real-time constraints. For these reasons, it was decided to use an AUTOSAR operating system for all parts with safety or real-time requirements.

Other requirements could not be immediately addressed with AUTOSAR, for instance the need to provide a graphical user interface with touch control and a TCP/IP based communication link via UMTS to the field study server and the gateway to the energy control center of Stadtwerke Trier and the charging infrastructure. Therefore, Linux was chosen as base for these functions.

Regarding the interaction between the two worlds of applications (AUTOSAR and Linux) the requirement was established to have a most flexible interface allowing for data communication with a high bandwidth and the invocation of services across OS boundaries. Last but not least, the chosen approach should be scalable to other system classes such as the combination of ADAS and IVI functionality as considered in this paper.

According to these requirements the prototype was realized as a hybrid system with AUTOSAR and Linux running on separate cores of the same microprocessor. The interaction between the system parts uses shared memory, thus allowing for the required flexible, high bandwidth interface.

3 Safety Challenges

The integration of IVI and ADAS functionality comes with challenges along more than one dimension. The focus of this paper is on the safety issues, therefore other aspects are largely ignored unless a clear relation to functional safety exists.

One problem of integrating safety-related functionality on IVI hardware is that hardware of this class might not be adequately qualified for safety-relevant issues. The automotive industry needs to monitor this issue and find suitable solutions. It is clearly beyond the scope of this paper to discuss this aspect further. However, the interested reader might have a look into a recent position paper related to this issue, that is part of an ongoing discussion on the usage of consumer electronics components in automotive applications [ZVEI14].

Another issue with such hardware is that it is typically designed to allow for a good average case performance and neglects the fact that worst case performance is of utmost importance for real-time systems. Typically this makes the precise prediction of worst case execution and reaction times of software on these systems extremely hard [Cul10]. Although several special aspects regarding real-time behavior are discussed throughout this paper, in-depth considerations of WCET analysis or schedulability analysis are out of the scope.

3.1 Observations on the ISO 26262

Part 9 of the ISO 26262 norm [ISO11] discusses „criteria for the co-existence of elements“. The considered cases are „coexistence within the same element of safety-related sub-elements with elements that have no ASIL assigned; and safety-related sub-elements that have different ASILs assigned“. Clearly this includes the case of IVI and ADAS software running on the same hardware. The norm prescribes that all sub-elements need to be developed to the highest ASIL level unless freedom from interference can be shown. As no Linux implementations are available that can be considered to satisfy ASIL A or higher, it is mandatory to show freedom from interference whenever ISO 26262 applies to such a combination.

The key term “freedom from interference” is often used in the context of combining safety-related and non-safety-related functionalities (or regarding combination of functionalities with different requirement classes on safety, e.g. ASIL). There are sometimes misconceptions about this term. First of all „freedom from interference“ does not mean that there is (or has to be) no influence between different classes of functionality. A sensible understanding could be that no undesired influence is allowed to be established. At least one would expect that harmful influence cannot occur. Surprisingly the definition used in the ISO 26262 norm is weaker than that.

Part 1 of the norm [ISO11] defines „freedom from interference“ as: „absence of cascading failures between two or more elements that could lead to the violation of a safety requirement“. Cascading failures are defined as follows according to the ISO 26262: „failure of an element of an item causing another element or elements of the same item to fail“ (cf. [ISO11], part 1). An example of a cascading failure would be, if one application crashes and thereby corrupts data needed by another application in a way that the latter also crashes. A difficulty arising from this definition is that it only captures situations where first one element fails and then one or more further elements also fail due to this first failure (failure: „termination of the ability of an element to perform a function as required“, cf. [ISO11], part 1).[2]

Consider the following example, Linux (or any other software element) starts and performs all functions as required. Additionally it writes into the memory area used by AUTOSAR (or of any other safety-related software element). In a strict sense this would still be understood as „freedom from interference“ according to the definition cited above. Certainly, it cannot be meant that way, because the affected element (e.g. AUTOSAR) could fail in an arbitrary way due to a fault in the causing element, but without the causing element failing by itself. We suggest that the definition of freedom from interference should also include this case of error propagation and only be used in the then stronger way of understanding. In the remainder of this paper we use freedom from interference in the stronger sense, i.e. including freedom from error propagation with subsequent failure. Please note, that unless expressed otherwise the terminology established in [Avi04] is used throughout this paper.

[2] The norm clearly distinguishes between “fault” and “failure” as in [Avi04]. Cascading failures according to the norm require one component to fail, it is not enough for the component to cause a fault.

In other parts the norm refers to another type of failures with importance for the considered case of integrating IVI and ADAS functionality: common cause failures. A common cause failure has a single root cause and affects two or more parts of a system, e.g. the AUTOSAR and the Linux part. The co-location of AUTOSAR and Linux on the same hardware clearly increases the potential for common cause failures. Potential root causes can be faults of the shared parts of the hardware, i.e. almost everything except for the processor cores. Moreover, the software responsible for booting the system could become a common cause for a failing start-up of both AUTOSAR and Linux in the current system design. The section on implementation gives details regarding the currently used boot sequence.

3.2 Relation between Safety and Security

For the considered combination of IVI and ADAS functionality on the same hardware it is not sufficient to focus on non-malicious faults, i.e. ignoring the potential actions of humans with the aim of causing harm. This includes the case of intruders on the Linux part of the system. Many solutions that are sufficient without malicious actions are rendered ineffective when human intruders need to be considered. Please note, that the current system relies on standard solutions to achieve security under Linux. This is not considered to be sufficient by the authors for the combination of IVI and ADAS systems. However, it is beyond the scope of the paper to discuss potential improvements in detail.

4 Current Implementation

The current prototype was previously presented in [NS13]. This section will now present a more detailed overview of how separation was achieved and which issues still remain open. The prototype was implemented on a Pandaboard ES (Revision B). The Pandaboard ES is an embedded development platform using the OMAP4460 SOC [TI14]. The OMAP4460 is based on the ARM architecture and comprises two Cortex-A9 cores, two Cortex-M3 cores as well as some other specialized cores for image processing, face detection etc. For the prototype only the two Cortex-A9 cores were used.

In addition to the Pandaboard, several peripheral hardware was added. To communicate with other ECUs, an MCP2515 CAN controller was connected via SPI. To track the location of the vehicle a GPS module was added, which is connected via UART. Both the GPS as well as the CAN module are controlled by AUTOSAR. For the user interface a display was connected over HDMI. This display is combined with a touch module, which is connected over USB. Finally, for communication a USB UMTS stick was used.

4.1 Boot Sequence

During start-up of a OMAP4460 the Cortex-A9 Core0 is initialized first by the ROM code. The ROM code fetches the boot code from non-volatile storage and starts executing it. The system uses Das U-Boot [De12] as a boot-loader to load an AUTOSAR

conforming operating system from non-volatile storage. This AUTOSAR image currently also includes a complete Linux kernel statically linked in a separate section of the executable. The additional space used up by the Linux kernel within the AUTOSAR image increases the load time from the non-volatile storage medium. We added additional start-up code which configures and starts the other core. On the second core a short start-up routine is used to load the Linux image contained in the AUTOSAR image and transfer control to the Linux image. All start-up parameters needed by Linux are provided by the core1 start-up routine and are written to the correct locations in memory. These parameters also include the `maxCPUs=1` option, which instructs the Linux kernel to run in single core mode.

There are some possibilities to further reduce the start-up times of AUTOSAR that were not currently taken. First, the load time of the AUTOSAR image with the added Linux image could be improved by adding a driver for the non-volatile storage and loading Linux on the second core. In this case, the load time for AUTOSAR would only be increased because of the added size due to the driver. Furthermore, Das U-Boot initialized several hardware modules that are not used by AUTOSAR, such as for example USB. Some of these hardware modules are later used without further re-initialization from within Linux, so that the initialization routines within Das U-Boot could not be removed. However, these initialization routines could be moved to the second core, so that the start-up of AUTOSAR is not delayed further by these hardware initializations. This additional initialization code could also be read from non-volatile storage on the second core, as not to increase the size of the AUTOSAR image.

4.2 Hardware Assignment

To reduce the possibility for interference between safety levels the hardware was assigned statically to one of the two cores, if possible. For shared hardware, it was necessary to protect all configurations made by AUTOSAR from changes through Linux and vice versa. Therefore, shared hardware was either initialized by the boot loader prior to starting AUTOSAR (e.g. main memory, interconnects) or the configuration routines in Linux were disabled or protected. Protection was implemented using the hardware locking mechanism provided by the OMAP4460 architecture [TI14]. The hardware assignment is shown in Table 1.

Hardware	Assigned to
Screen	Linux
Touch-pad	Linux
UMTS module	Linux
USB subsystem	Linux
Non-volatile memory (SD-Card)	Linux
Main Memory (DRAM)	Linux/AUTOSAR (configured by u-boot)
L1 Caches	One per Core (only activated in Linux)
L2 Cache	Linux

L3 OCM RAM (SRAM)	AUTOSAR
CAN module	AUTOSAR
SPI interface	AUTOSAR
UART interface	AUTOSAR
L3 and L4 interconnects	Linux/AUTOSAR (configured by u-boot)
Interrupt Distributor	Linux/AUTOAR (protected using HW spinlocks)
Interrupt CPU Interfaces	One per OS
Hardware spinlocks	Linux/AUTOSAR (configured by AUTOSAR)
Clock Tree	Linux/AUTOSAR (configured by u-boot and managed by Linux)

Table 1: Hardware Assignment (taken with additions from [NS13]).

4.3 Memory

The used platform provides several types of memory, which were used: Non-volatile memory (SD-Card), 1GB of DRAM, 1Mb L2 cache shared between cores, 32kB L1 data cache per core, 32kB L1 instruction cache per core and 56kB SRAM. The non-volatile RAM was fully assigned to and managed by Linux.

Within DRAM a special area was reserved for AUTOSAR. The physical address range used for this area was reserved within Linux using the `mem=` kernel command line parameter. Using this parameter the reserved range is by default excluded from any mappings in the MMU. However, additional mappings to this memory region can be added manually within the kernel. Furthermore, this memory may be accessed using the `/dev/mem` device node with root privileges. Therefore, the AUTOSAR memory region was protected from unintended accesses using the MMU. Security flaws within Linux that provide elevated privileges or access to kernel space may still be abused to tamper with the AUTOSAR memory region. The reserved memory region is also used for communication facilities. For this a part of the memory region is mapped from a kernel module as IO memory. This shared memory can be used to implement the data structures necessary for communication. For AUTOSAR the reserved section of the DRAM was used for code and global data.

To ensure predictability the MMU was deactivated by AUTOSAR for its own core. By deactivating the MMU, unpredictable delays from unexpected TLB refills were avoided [Sch12b]. Also, for OMAP4460 SOCs disabling the MMU also disables all caches for memory used by that core. This way interferences between Linux and AUTOSAR from sharing the L2 cache and cache coherency protocols are avoided. However, disabling the caches also greatly reduces the performance of the system. As a first step to mitigate this problem, the SRAM section was completely assigned to AUTOSAR and disabled in Linux. In our final implementation, all stacks were moved to the SRAM to increase

speed[3]. While this method provides some speed improvements, future modifications can further improve the usage of SRAM for speedup. Also instead of statically assigning parts of the system to SRAM it may be possible to use the SRAM as a scratchpad memory managed by the OS or the compiler [ABS01]. Furthermore, freedom from interference only requires disabling the shared L2 cache. Depending on the architecture, it may be possible to only enable the unshared L1 cache [ARM11a]. Assigning the SRAM section to AUTOSAR, as in the current implementation, requires that the SRAM is not used by Linux. Currently SRAM may be accessed within Linux using a kernel module. To test if the memory could be safely assigned, this module was instrumented with log outputs. These outputs showed that SRAM was not used by the Linux portion.

The current solution leaves two main issues. First, using the `mem=` parameter the address space of AUTOSAR is excluded from the memory mapping used by Linux. However, there is no restriction within Linux preventing it from changing this memory mapping. Erroneous code within the kernel or with root privileges could in principle still map any part of the AUTOSAR address space, hence impairing safety requirements. For this reason all parts of the user interface were run without root privileges if possible. Secondly, disabling the caches within AUTOSAR will greatly decrease the speed. While assigning the SRAM to AUTOSAR may mitigate this problem somewhat, this only partially solves the issues. A better solution would be to enable L1 caches globally and also analyze for which portions of the code or data the L2 caches could be activated as well.

4.4 Interrupts

The OMAP4460 SOC uses an ARM generic interrupt controller (GIC) [ARM11b] to configure and deliver interrupts to the cores. The GIC is composed of a single shared interrupt distributor and one CPU interface per core. The interrupt distributor receives interrupt signals and distributes them to the CPU interfaces for handling by the core. Interrupt signals can be distributed to one or more cores [ARM11b].

In the current state of the system, interrupts are statically assigned to one of the two cores. During start-up AUTOSAR configures all interrupt signals required by itself to be targeted to the core on which it is running. The initialization routine for the GIC within Linux has been changed not to reassign any interrupts targeted for the AUTOSAR core. Additional care has been taken to avoid race conditions. For the ARM GIC used by the OMAP4460 architecture interrupt targets are configured by using memory mapped registers. For these registers one processor word is used to manage the targets for four different interrupts. In Linux the mapped words are written using a read-modify-write pattern. To avoid races with AUTOSAR these registers are protected using a HW lock[4].

[3] For the version of the system presented in [NS13] all data portions were put into SRAM. Later changes to the system however caused the bss section to grow beyond 56kB so that this section was removed from SRAM.
[4] The ARM GIC specification [ARM11b] indicates that these registers are also byte accessible. However the Linux kernel currently does not use this method to set the targets, so that races are possible if both AUTOSAR and Linux try to modify the same register and no additional locking is used.

If it can be guaranteed that all interrupts are completely initialized within AUTOSAR prior to starting Linux these locks are not needed. In that case, it can also be guaranteed that the shared hardware does not introduce any additional delays. Similarly, interrupt priorities are managed by using one word to store the priorities of four different interrupts. Again, these registers must be protected using HW locks shared between AUTOSAR and Linux or it must be ensured that all priorities are set by AUTOSAR prior to loading Linux and are not changed while the system is running.

Masking interrupts can be done in multiple ways using the ARM GIC. First, all interrupts can be disabled for a single core using the `cpsid` instruction. Second, some portion of the interrupts can be masked for a single core using priority masking in the CPU interface for that core. Third, single interrupts can be globally masked using the distributor. Currently within the used AUTOSAR implementation interrupts are only disabled via the `cpsid` instruction. No masking of single interrupts or groups of interrupts is done. However, these functionalities could easily be used. Masking based on interrupt priorities can be done using the CPU interface, so that no shared hardware is needed. Also, the registers of the distributor for masking/unmasking of single interrupts are implemented race-free by using separate set and clear registers. This means no additional locks are needed for performing masking of interrupts independently on both systems.

Using the current solution there is no guarantee that Linux will not deactivate any interrupts targeted towards AUTOSAR either by masking them or by setting another target core. However like any hardware, the GIC registers are protected within Linux by the MMU. The code which maps these registers to configure the GIC is very small and could be easily verified. If this code is instrumented so that it will not change any interrupts targeted towards AUTOSAR it would be possible to ensure that this portion of the Linux kernel does not cause any failures of safety-related functions. This, however, still leaves the possibilities of other parts of the Linux kernel mapping the GIC registers and changing the configuration of the interrupts. This could either be done by an attacker or because of severe bugs in the Kernel. One possibility to mitigate this issue would be to secure any access to the MMU, so that no additional mappings to the GIC registers can be introduced.

4.5 Interconnects

The OMAP 4460 SOC uses several types of interconnects for communication between hardware components. Both Cortex-A9 cores are connected to most other SOC hardware by a local interconnect. This local interconnect further connects to the L3 interconnect and the L4_ABE interconnect. The L4_ABE interconnect serves hardware, which is part of the audio backend. Other hardware, which is used (UART, SPI, some timers) is served by the L4_PER interconnect, which in turn is connected to the L3 interconnect [TI14].

Since the system only provides a single set of interconnects for both cores, these interconnects have to be shared between Linux and AUTOSAR. This also means, that

requests to configure or service hardware from Linux may delay requests from AUTOSAR. The L3 interconnect uses a leaky bucket algorithm [Tu86] for bandwidth regulation [TI14]. Using this algorithm the bandwidth for each master on the interconnect is limited and data can be processed at a higher priority as long as the maximum configured bandwidth is not exceeded. However, since both cores in the Cortex-A9 dual processor are connected to the L3 interconnect using the same local processor interconnect, the bandwidth for both cores can only be set together. Still, this method may be used to protect accesses to safety relevant hardware by the processor, to guarantee freedom from interference from other hardware.

Memory is connected directly to the cache and memory controllers, which are part of the dual core Cortex-A9 subsytem [TI14]. Thus, memory read or write accesses from the core that is running AUTOSAR only need to be arbitrated with other memory accesses. No arbitration is needed for accesses to memory and accesses to other hardware from Linux.

Currently AUTOSAR also uses two timers, which are connected using the L3 and L4_PER interconnects. These timers are used for system ticks and timeouts of the UART module. In case of heavy congestion on one of the two interconnects, configuration updates to these timers could be delayed. As a possible solution timers which are part of the ABE module could also be used. These timers are fully accessible from the Cortex-A9 cores using the L4_ABE interconnect only. If these timers are used and the ABE is disabled within Linux, no congestion can appear on the L4_ABE interconnect.

As discussed congestion may be an issue, which can influence real-time behavior. However, most important hardware, e.g. memory, is connected separately to the cores thus limiting the potential effects. Furthermore, depending on the hardware it may be possible to use reserved interconnects to avoid any congestion. For example, in case of the OMAP4460 SOC timers within the ABE backend could be used [TI14]. While this may require deactivating some part of the hardware, this may be easily replaceable for example using a soundcard connected via USB.

4.6 Clock-Tree and Power Management

Currently clock-tree and power management are performed by Linux. The complete clock-tree is initialized during start-up by U-boot and later managed by Linux. Experiments on the hardware platform showed a risk of overheating during full load, if the speed is not throttled. If code on the Linux side can cause the SOC to fail due to overheating, this could also severely impact safety-related functions. Hence, a thermal management is needed for reliability. This thermal management is currently provided by the Linux kernel. Unfortunately, the CPU speed can only be set for both cores together, so that Linux will also control the speed of AUTOSAR. To ensure that all deadlines are met, timing analysis on AUTOSAR was done at the lowest possible rate. However, the uncontrolled change of clock speed can cause serious problems in real-time software, even if scheduling and timing anomalies could be avoided. Therefore, the control of the clock speed should be given to the safety-related real-time part, i.e. AUTOSAR in the case of combining IVI and ADAS functionality. Moreover the current implementation

limits the processing capacity that can be used for safety-related functions to the capacity provided by the chip at the lowest frequency. A closer analysis which frequencies do not cause thermal issues could also allow running the system at a higher speed. In that case all lower frequencies should be disabled within Linux.

It would be possible to implement the thermal management within AUTOSAR. With such an implementation, either power management could be done by AUTOSAR independently without regards for Linux. If the Linux programs can run at arbitrary speed, this is the simplest solution. On the other hand, it would still be possible to use the normal power management infrastructure within Linux and forward the decision of the installed CPUfreq governor to AUTOSAR. AUTOSAR could then use this additional information to provide the needed CPU speed to Linux.

Power management is a severe issue for any system using multiple operating systems. Throttling the CPU is only possible in cooperation with both systems. Instead of throttling the usual solution is to temporarily completely halt cores when possible [Chi08]. This method is also employed by the current prototype. Both Linux and AUTOSAR perform a `wfi` instruction in their idle tasks, which reduces the power consumption of the current core. Still thermal issues may require more sophisticated solutions to the problem of power management. Without any thermal management depending on the hardware an additional point of failure or attack would be opened. For example, an attacker could just run a program which burns CPU cycles, thus increasing the heat, which may lead to a failure of both cores. Such an attack might also be run without any elevated privileges. Similarly, faulty code could also burn CPU cycles and increase the heat on the die. This means that any solution in which safety-related functions are combined with non-safety-related ones on the same hardware requires a thermal management which can be verified not to lead to any failures. This also applies to solutions in which there is a partitioning hypervisor.

4.7 Communication

To share data between AUTOSAR and Linux real-time communication facilities were added. For stream transmissions a non-blocking ring buffer implementation was used. Multiple ring buffers are provided for multiple data streams. Using these ring buffers AUTOSAR tasks can send data packets to Linux, which are then received in a kernel thread (bouncer thread). This kernel thread takes the packet and copies it to a lookaside cache, which is created using the `kmem_cache_create()` method [CKR05]. Lookaside caches are frequently used within Linux to flush buffers used by hardware, e.g. network cards, to internal kernel buffers. The `kmem_cache_create()` provides a pool allocator for blocks of fixed size. Blocks which are freed are not directly returned to the free memory managed by the kernel, but rather to the pool for quick re-use. Pools for blocks of equal size can be shared. The bouncer thread is scheduled every 100ms and removes all packets from the ring buffer to the lookaside cache so that the ring buffer itself remains usable from within AUTOSAR, even if the packets are not picked up by a user-space process. All packets are stored in a linked list, which can be retrieved using a device node. In AUTOSAR multiple tasks sending data simultaneously through the same

ring buffer must be synchronized with each other. For this the real-time resource sharing mechanisms within AUTOSAR are used, so that real time capabilities are not impaired. The structure of the communication facilities is presented in Figure 1.

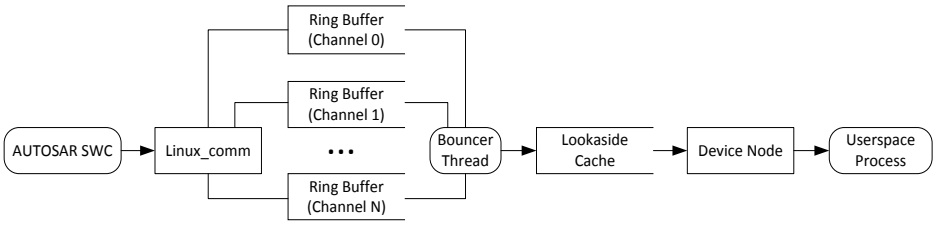


Figure 1: The structure of the communication facilities from AUTOSAR to Linux

In case the lookaside cache grows beyond a configurable limit, no further packets are retrieved from the ring buffers by the bouncer thread. In this case the ring buffers may overflow if further packets are written on the AUTOSAR side. Similarly, if the kernel bouncer thread is not regularly scheduled, this may also cause congestion in one of the ring buffers. If a ring buffer overflows the packet is dropped within AUTOSAR to enforce the non-blocking behavior. This may result in loss of data on the side of Linux, but not in any missed deadlines on the AUTOSAR side. As Linux is not supposed to perform any safety relevant functions, the missed data does not impair the overall safety of the system.

To ensure no delays during communication from caches or cache coherency protocols, caching for the shared memory areas is disabled in Linux. Caches are deactivated by mapping the memory area as IO memory within the kernel. No caches are used within AUTOSAR. To keep the pointers indicating begin and end of the ring buffer consistent, data is first written to the buffer within AUTOSAR, then a memory fence is issued and after the fence the pointer is updated. Similarly in Linux first the data is read and then after a memory fence the pointer is updated.

Although this ring buffer mechanism is only used to send data from AUTOSAR to Linux, the opposite direction could be supported as easily. As a secondary communication mechanism for state data (as opposed to stream data), atomic writes to reserved memory words are used. This communication mechanism is currently only used to communicate data from Linux to AUTOSAR.

While the current communication structure could still be improved, for example to decrease the risk of lost data, it does not pose any risks to safety-related functions. If any of the buffers overflow data may be lost. However, since this data is accepted by Linux, this transmission must not be part of any safety-related functions. Similarly, the data that is received from Linux must be treated as unreliable by all safety-related functions. Hence, even if the communication mechanism misbehaves in any way, the safety-related functions cannot be impaired in any way.

5 Competing Solutions

The usual solution to ensure freedom from interference is to use separate hardware for safety-related and non-safety-related functions. However, separating the hardware increases the cost of the system. Also, separating the hardware requires additional communication facilities such as dedicated busses. Moreover, the required flexible and high bandwidth communication channel is not available in such a setting. This impedes innovations that utilize a close coupling of IVI and ADAS functionality.

Another possibility is to develop the complete system according to the highest ASIL. However, this requires extensive verification and testing of non-safety-related functions, which may not be feasible. This would also have to encompass the complete Linux kernel, if a Linux portion is used. While some soft real-time adaptations of the Linux kernel exists [RH07] these have not been developed according to ISO 26262 and hence cannot be employed for safety-related functions. Using different operating systems than Linux typically prevents short development cycles and increases development costs.

A third possibility is to use a hypervisor to virtualize the two systems [ISM09]. However, the additional software layer will have an impact on performance [Bru10,Sch12a] that may not be acceptable in all cases. Also, to use a hypervisor for partitioning of safety-related and non-safety-related functions, it is necessary to verify the hypervisor at the level required by the safety-related functions. Furthermore, it must be possible to guarantee that the scheduling of the hypervisor can ensure real-time performance for the safety-related system parts. This may for example require additional idle time allocated to the real time portions. This further reduces the possible performance.

References

[ABS01] Avissar, O.; Barua, R.; Stewart, D.: Heterogeneous memory management for embedded systems. In Proceedings of the 2001 international conference on Compilers, architecture, and synthesis for embedded systems. ACM, 2001.

[ARM11a] Cortex-A Series - Programer's Guide. Version 2. ARM, 2011.

[ARM11b] ARM Generic Interrupt Controller – Architecture Specification. Architecture Version 2. ARM, 2011.

[Avi04] Avizienis, A.; Laprie, J. C.; Randell, B.; Landwehr, C.: Basic concepts and taxonomy of dependable and secure computing. In IEEE Transactions on Dependable and Secure Computing, 2004, 1, 11-33.

[Bru10] Bruns, F.; Traboulsi, S.; Szczesny, D.; Gonzalez, E.; Xu, Y.; Bilgic, A.: An evaluation of microkernel-based virtualization for embedded real-time systems. In 22nd Euromicro Conference on Real-Time Systems (ECRTS), 2010.

[Chi08] Chisnall, D.: The Definitive Guide to the XEN Hypervisor. Prentice Hall, 2008.

[CKR05] Corbet, J.; Kroah-Hartman, G.; Rubini, A.: Linux Device Drivers, 3rd Edition. O'Reilly, 2005.

[Cul10] Cullmann, C.; Ferdinand, C.; Gebhard, G.; Grund, D.; Maiza, C.; Reineke, J.; Triquet, B.; Wilhelm, R: Predictability considerations in the design of multi-core embedded systems. In Proceedings of Embedded Real Time Software and Systems (2010): 36-42.

[De12] Denk, W: Das U-Boot. 2012. url: http://git.denx.de/cgi- bin/gitweb.cgi?p=u-boot.git;a=summary.

[ISM09] Iqbal, A.; Sadeque, N.; Mutia, R. I.: An overview of microkernel, hypervisor and microvisor virtualization approaches for embedded systems. Report, Department of Electrical and Information Technology, Lund University, Sweden 2110 (2009).

[ISO11] Road vehicles – Functional safety. ISO 26262, First Edition. International Organization for Standardization, 2011.

[KT05] Kempton, W.; Tomić, J.: Vehicle-to-grid power implementation: From stabilizing the grid to supporting large-scale renewable energy. In Journal of Power Sources, vol. 144, no. 1, pp. 280–294, 2005.

[NS13] Nett, T.; Schneider, J.: Running Linux and AUTOSAR side by side. In 7th Junior Researcher Workshop on Real-Time Computing, pp. 29-32, Sophia Antipolis, France, 2013.

[NS14] Nett, T.; Schneider, J.: Automated Planning of Charge Processes for Privately Owned Electric Vehicles. In 3rd International Conference on Connected Vehicles & Expo (ICCVE), 2014.

[RH07] Rostedt, S.; v. Hart, D.: Internals of the RT Patch. In Proceedings of the Ottawa Linux Symposium. Vol. Two. Ottawa, Canada, 2007, pp. 161–171.

[Sch12a] Schneider, J.: Overcoming the Interoperability Barrier in Mixed-Criticality Systems. In Proceedings of the 19th ISPE International Conference on Concurrent Engineering – CE, 2012

[Sch12b] Schneider, J.: Why current Memory Management Units are not suited for Automotive ECUs.In Proceedings of the Automotive Safety & Security, 2012.

[TI14] OMAP4460 Multimedia Device Silicon Revision 1.x. Technical Reference Manual. Version AB. Texas Instruments, 2011 (revised 2014).

[Tu86] Turner, J.: New directions in communications (or which way to the information age?). In IEEE Communications Magazine, Volume 24, Issue 10, 1986

[ZVEI14] Position Paper: Consumer Components in Safe Automotive Applications. German Electrical and Electronic Manufacturers' Association (ZVEI e.V.), July 2014, http://www.zvei.org/Publikationen/Position-paper-Consumer-Semi-Automotive.pdf.

Umsetzung der Anforderungen aus der ISO 26262 bei der Entwicklung eines Steuergeräts aus dem Fahrerinformationsbereich.

Dr. Antje Gieraths[1], Christoph Müller-Albrecht[1], Sean Brown[2]

[1]Instruments & Displays, HMI Software
BMW Group
[2]Telemotive AG
antje.gieraths@bmw.de
christoph.mueller-albrecht@bmw.de
sean.brown@partner.bmw.de

Kurzfassung: Die seit November 2011 geltende ISO Norm 26262 gilt als die Norm zur Erreichung und Umsetzung funktionaler Sicherheit im Automobil. In diesem Artikel wird anhand eines konkreten Steuergeräts beschrieben wie die Anforderungen aus der ISO Norm für die ASIL Abstufungen A und B erreicht werden können. Das gewählte Steuergerät beeinhaltet dabei Sicherheitsanforderungen und Anforderungen aus dem Bereich der Fahrerinformationssysteme, die in Einklang gebracht werden müssen. Die ISO Norm gibt verschiedene Arbeitsschritte vor, beispielsweise die Definition der Anforderungen, die Implementierung der Software und die verschiedenen Testarten. In diesem Artikel soll besonders auf den Entwicklungsschritt der Softwarearchitekturdefinition eingegangen werden. Es wird gezeigt, wie die in der ISO Norm geforderte „Freedom from Interference“ durch Software Design umgesetzt werden kann. Die hierfür notwendigen Voraussetzungen, z.B. auf Betriebssystemseite werden erläutert. Insbesondere im Sicherheitsbereich spielt eine hohe Qualität der Software und eine kontinuierliche Rückmeldung darüber an den Projektleiter und die Entwickler ein wichtige Rolle. Entsprechende Metriken sind während der Softwareentwicklungsphase kontinuierlich zu erheben. In diesem Beitrag soll gezeigt werden wie Continuous Integration mit einem hohen Automatisierungsgrad zur Umsetzung der ISO Norm eingesetzt werden kann. Dabei wird auf die konkreten Implementierungsschritte und einige Werkzeuge eingegangen.

1 Einleitung

1.1 Prämissen

Seit November 2011 ist die ISO26262 in Kraft. Sie gilt als die Norm zur Erreichung und Umsetzung funktionaler Sicherheit im Automobilbereich. Sie leitet sich aus der IEC EN DIN 61508 ab. Die ISO26262 gibt ein Vorgehensmodell mit standardisierten

Arbeitsprodukten vor. Beispiele für die Arbeitsprodukte sind das technische Sicherheitskonzept, der Verification Plan oder der Verification Report. Die ISO Norm deckt die Konzeptphase, die Produktentwicklungsphase, die Produktion, die Prozesse sowie die Analysen als Arbeitsschritte ab. Für ein Steuergerät wird die Produktentwicklung auf Systemebene, auf Hardware und Software Ebene (Teil 5 und 6 der ISO Norm) beschrieben. Die ISO Norm beschreibt die Umsetzung der funktionalen Sicherheit für vier Abstufungen der Automotive Safety Integrity Level (ASIL) – A bis D. In diesem Artikel wird auf das ein Steuergerät aus dem Fahrerinformationsbereich eingegangen, das die Level A bzw. B zu erfüllen hat. Ausgehend von einem Steuergerät, das als Zwei-Prozessorsystem aufgebaut ist und das Headup-Display (HUD) sowie das Kombiinstrument ansteuert, wird erläutert, wie die Darstellung von sicherheitsrelevanten Anzeigen wie den Check Control Meldungen (z.B. „Motorhaube offen“) umgesetzt wird.

1.2 Motivation

Aus anderen Anforderungen, die beispielsweise die passive und aktive Sicherheit beim Fußgängerschutz betreffen, entstehen Entwicklungen wie eine elektronische Verriegelung der Motorhaube im Fahrzeug. So kann bei einem Zusammenprall mit einem Fußgänger oder Radfahrer die Motorhaube aufgestellt werden. Die Schwere der (Kopf-) Verletzungen kann durch diese Maßnahme gemindert werden. Zur Umsetzung dieser Maßnahme ist es jedoch erforderlich, dass die Motorhaube geschlossen ist und ein Fehler dem Fahrer zur Anzeige gebracht wird. Hier findet der Austausch einer mechanischen durch eine elektonische Komponente statt. Dies hat jedoch zur Folge, dass daraus Anforderungen an Anzeigeinstrumente zur ISO26262-konformen Software Entwicklung entstehen. Das heißt, es muss eine Anzeige geben, die den Fahrer über den Zustand der elektronischen Motorhaubenverriegelung sicher und zuverlässig informiert (Check Control „Motorhaube offen.“, siehe Abbildung 1).

Abbildung 1 - Motorhaube offen - Checkcontrol Meldung im freiprogrammierbaren Kombiinstrument.

2 Vorgehensmodell

Die ISO Norm gibt ein V-Modell zur Umsetzung der Software Anforderungen vor. Es beginnt mit dem Systementwurf („Initiation of Product Development“) und der Spezifikation der Sicherheitsanforderungen. Jeder Abschnitt in der ISO Norm enthält nach den ASIL Leveln gestaffelte Maßnahmen. Diejenigen Schritte, die mit „++“ bzw. highly recommended gekennzeichnet sind, sind als verpflichtend für das entsprechende ASIL Level anzusehen. Zu den ersten Schritten gehört beispielsweise auch die Definition von Modellierungs- und Codierungsrichtlinien (vgl. Abbildung 2). Für jeden Prozessschritt entstehen so konkrete Anforderungen, die überprüfbar und mit Hilfe von Werkzeugen umgesetzt werden können.

Table 1 — Topics to be covered by modelling and coding guidelines

	Topics	ASIL			
		A	B	C	D
1a	Enforcement of low complexity[a]	++	++	++	++
1b	Use of language subsets[b]	++	++	++	++
1c	Enforcement of strong typing[c]	++	++	++	++
1d	Use of defensive implementation techniques	+	+	++	++
1e	Use of established design principles	+	+	+	++
1f	Use of unambiguous graphical representation	+	++	++	++
1g	Use of style guides	+	++	++	++
1h	Use of naming conventions	++	++	++	++

[a] An appropriate compromise of this topic with other methods in this part of ISO 26262 may be required.

[b] The objectives of method 1b are
- Exclusion of ambiguously defined language constructs which may be interpreted differently by different modellers, programmers, code generators or compilers.
- Exclusion of language constructs which from experience easily lead to mistakes, for example assignments in conditions or identical naming of local and global variables.
- Exclusion of language constructs which could result in unhandled run-time errors.

[c] The objective of method 1c is to impose principles of strong typing where these are not inherent in the language.

Abbildung 2 Anforderungen an die Modellierungsrichtlinien aus dem Teil 6 der Software Anforderungen. Quelle[ISO26262]

3 Umsetzung von Architekturanforderungen

3.1. Signalfluss im Kombiinstrument:

Das als Kombiinstrument bezeichnete Steuergerät besteht aus zwei Prozessoren – der automotive Prozessor (MCU) und dem Grafikprozessor (GPU). Das Steuergerät wird bei BMW als big-little System entworfen, d.h. man geht von einem relativ kleinen automotive Kern aus und einem großen Grafikprozessor, der seinerseits einen Grafikkern und eine kleine CPU enthält. Auf der MCU läuft AUTOSAR als Betriebssystem und verschiedene Software Komponenten, beispielsweise

Signalverarbeitungsbibliotheken und Treibersoftware. Des weiteren gibt es Komponenten, die das Flashen und die Diagnosefähigkeit umsetzen. Abbildung 3 zeigt einen groben Überblick über den Signalfluss innerhalb der Software. Ein Signal, z.B. das der Ganganzeige kommt über den CAN Bus an MCU an, durchläuft die verschiedenen AUTOSAR Schichten und gelangt dann zu den Signalverarbeitungsbibliotheken, dort wird das Signal gefiltert (wenn notwendig) und auf Fehler überprüft. Die Kommunikation zwischen den Modulen findet über die AUTOSAR RTE (Run Time Environment) statt. Die Treiber, die beispielsweise das Backlight des Displays ansteuern, tauschen ihre Daten mit anderen Software Komponenten auf der MCU über die RTE aus. Das Signal, das ursprünglich über den CAN auf dem Steuergerät ankam, gelangt dann über die Interprozessor Kommunikation auf die GPU. Dort wird es von der HMI (Human Machine Interface) Software dargestellt.

3.2 Umsetzung

In [ISO26262], Teil 6, Abschnitt 4 heißt es:

When claiming compliance with ISO 26262, each requirement shall be complied with, unless one of the following applies:

a) tailoring of the safety activities in accordance with ISO 26262-2 has been planned and shows that the requirement does not apply, or

b) a rationale is available that the non-compliance is acceptable and the rationale has been assessed in accordance with ISO 26262-2.

Das bedeutet, es gibt drei Möglichkeiten die Anforderungen der ISO Norm zu erfüllen:

1. Alle Software ist nach dem geforderten ASIL Level zu entwickeln.
2. Alle Software wird nach dem geforderten ASIL Level entwickelt und es werden Argumentationen und Begründungen gegeben, die nach der ISO Norm akzeptiert werden.
3. Die Software wird so partionioniert, dass nur die Teile, die für die funktionale Sicherheit relevant sind, nach ASIL qualifiziert werden und die sogenannte „Freedom from Interference“ , d.h. Rückwirkungsfreiheit gewährleistet ist.

Die Software Architektur, in der die Komponenten und ihre Funktionalität und ihr Zusammenspiel mithilfe dynamischer und statischer Sichten festgelegt wird, ist also von immenser Wichtigkeit bei der Umsetzung der ISO Norm [BS2012]. Die Methode der „Decomposition“ bei der Betrachtung einer Komponente als „Safety Element out of Context“ kann sehr hilfreich sein. Hier kann ein System so entworfen werden, dass aus zwei Komponenten, die ASIL A erfüllen, Anforderungen nach ASIL B erfüllt werden können – durch geschicktes Systemdesign und eine entsprechende Aufteilung.

Der in 3.1 beschriebene Signalfluss muss nun nach den in der ISO Norm beschriebenen Kriterien abgesichert werden. Auf der MCU bedeutet dies ein entsprechendes Betriebssystem – AUTOSAR mit ASIL B einzusetzen, das Speicherschutz und Laufzeitgarantie bietet. Über CRC[1] Checks und Alive Counter wird die Aktualität und Validität des Signals überprüft. Dies gilt ebenfalls für die Übertragung zur GPU.

In der HMI Software sollen nun verschiedene Anzeigen mit der ASIL Einstuftung A integriert mit anderen Anzeigen (Tacho, Drehzahlmesser, etc.) ohne ASIL Einstufung angezeigt werden.

Als Prämisse für die vorgestellte Lösung gilt außerdem, dass der verwendete Grafikprozessor mehrere Hardware Grafiklayer zur Verfügung stellt. Dies ist notwendig um sicherzustellen, dass die sicherheitsrelevanten Anzeigen auf einem separaten Layer angezeigt werden können. Dieser muss auch so konfiguriert werden können, dass er nicht überschrieben werden kann.

Im folgenden wird beschrieben, wie das Konzept der *Freedom from Interference* auf der GPU Seite umgesetzt werden kann.

Als zentrale Idee werden sicherheitsrelevante und nicht-sicherheitsrelevante Software Anteile von einander getrennt, damit sie sich nicht negativ beeinflussen können. Es wird eine sogenannte SafetyHMI entwickelt, die nur die ASIL-relevanten Anzeigen enthält. Die anderen Anzeigen werden in der regulären HMI Software zusammengefasst und müssen die Sicherheitsanforderungen nicht erfüllen. Dazu ist ein Betriebssystem notwendig, welches Laufzeit- und Speicherschutz bietet. Die Inhalte für die funktionale Sicherheit befinden sich in einem eigenen (virtuellen) Adressraum und werden über einen ebenfalls nach ISO26262 entwickelten Grafiktreiber angezeigt. Durch die verschiedenen Grafiklayer, welche die GPU Hardware bereitstellt, ist es möglich sicherheitsrelevante Anzeigen auf einem Layer zu konzentrieren. Die Layer werden dabei so konfiguriert, dass der oberste Layer mit seinen Elementen immer sichtbar bleibt.

Abbildung 3 zeigt den Aufbau der „Safety HMI“ und der HMI innerhalb des Gesamtaufbaus. Um die Komplexität in der SafetyHMI so gering wie möglich zu halten, werden Texte als vorgerenderte Bilder im Speicher gehalten.

Wenn die Safety HMI nach allen Regeln der ISO Norm entwickelt wird, dann können mit der oben genannten Trennung alle sicherheitsrelevanten Anzeigen abgedeckt und die Konformität mit der ISO26262 sichergestellt werden.

[1] CRC = Cyclic Redundancy Check

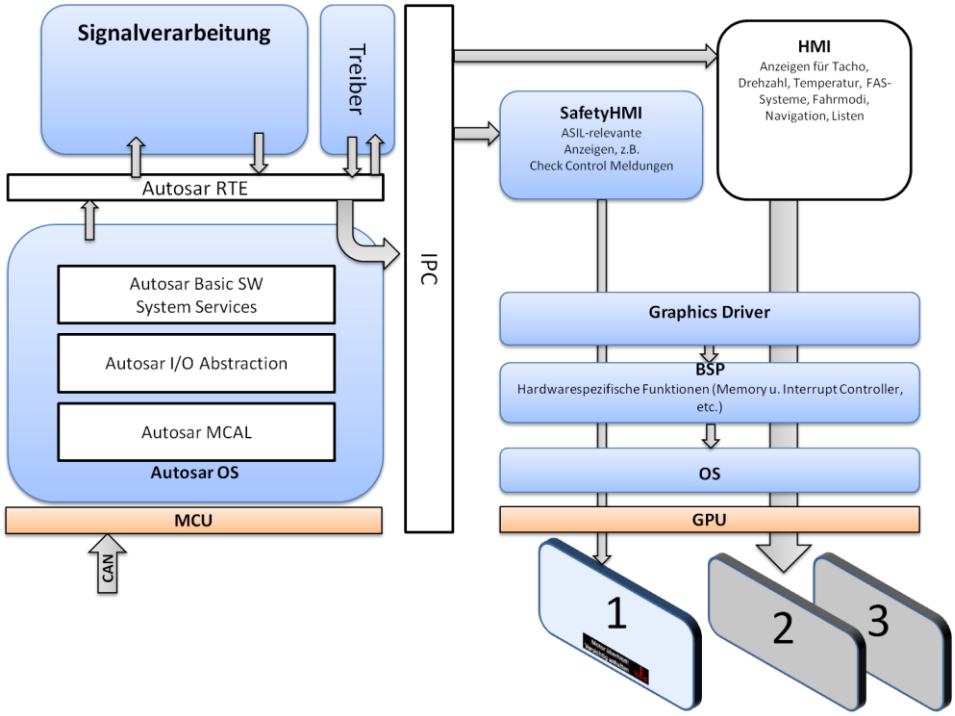


Abbildung 3 Signalfluss Diagramm mit SafetyHMI und HMI. Die sicherheitsrelevanten Anzeigen werden immer im vordersten Layer angezeigt. Die beiden HMI Softwaren laufen parallel, aber in verschiedenen Adressräumen. Alle hellblau markierten Flächen müssen nach der ISO Norm 26262 entwickelt werden. Die Betriebssysteme auf dem Grafikcore und dem Automotive Core müssen Laufzeitgarantie und Speicherschutz bieten. Der erste Layer ist der unüberschreibbare Hardware Layer, welcher die sicherheitsrelevante Anzeigen darstellt. Der zweite Layer dient zur Darstellung der regulären HMI Software mit Tachometer, Drehzahlmesser, den Fahrmodi, der Navigation und den Listen. Der dritte Layer stellte die Anzeigeelemente auf dem Head Up Display (HUD) dar.

4 Continuous Integration zur Umsetzung der ISO Norm.

Continuous Integration und auch Continuous Deployment sind Schlagwörter, die in den letzten Jahren zunehmend an Bedeutung gewonnen haben [QT2012]. In der Software Entwicklung für eingebetteten Systeme sind schnelle und flexible Entwicklungszyklen mit einer raschen Rückmeldung an die Entwickler genau so wünschenswert wie im Bereich der Websoftware.

Zur Umsetzung der ISO26262 müssen u.a. die folgenden drei Abschnitte in der Softwareentwicklung durchgeführt werden:

1. Software Design und Implementierung

2. Software Modul Test

3. Software Integration und Test

In diesem Abschnitt soll darauf eingegangen werden, wie die Anforderungen, die sich aus der ISO26262 ergeben, mit Continuous Integration nachhaltig erfüllt und welche Tools dabei genutzt werden können.

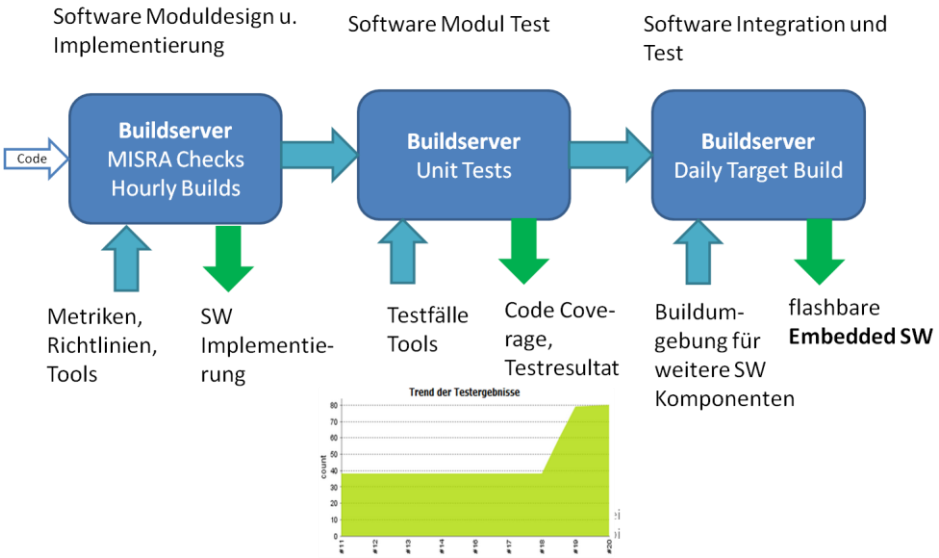


Abbildung 4 Buildkette für die Erstellung der eingebetteten Software. Als Ergebnisse der einzelnen Schritte erhält man die implementierte Software, die Testresultate und die Code Coverage sowie aus dem letzten Schritt die flashbare Software für die Target Hardware.

4.1 Software Design und Implementierung

In diesem Schritt werden die Software Module erstellt und die Funktionalität implementiert. Es gilt nach vorher festgelegten Regeln zu programmieren und die Einhaltung dieser Regeln messbar zu überprüfen.

Die Software muss kompilierbar sein und auf eventuelle Fehler der Entwickler muss schnell reagiert werden können.

Eine Continuous Integration Umgebung, beispielsweise der Jenkins Server [Jenkins] wird dazu genutzt, mehrfach pro Tag, idealerweise nach jeder Veränderung des Quellcodes in der Versionsverwaltung die Software zu kompilieren. Parallel geben Werkzeuge zur statischen Code Analyse, die automatisiert auf dem Server gestartet werden, den Entwicklern und dem Projektleiter eine Rückmeldung bezüglich der Verletzung von vorher festgelegten Programmierrichtlinien. Auch diverse Metriken, wie die Anzahl der Codezeilen, die zyklomatische Zahl oder die Kommentarfrequenz können

so erfasst werden. Bei Verletzung der Regeln erhalten die Entwickler direkt Rückmeldung über den Server.

Als Ergebnis dieses Schrittes der Continuous Integration Kette erhält man die implementierte und geprüfte Software (vgl. Abbildung 4).

4.2. Software Modul Test

Eine ausreichende Testabdeckung ist essentiell zur Erreichung einer zufriedenstellenden Qualität und vor allem Stabilität im Software Entwicklungsprozess. Auch hier kann Continuous Integration unterstützen. Die Testfälle und entsprechende Werkzeuge, wie das entsprechende Testframework (CUnit, Googletest, etc.) laufen automatisiert auf dem Buildserver. Auch hier erhalten Entwickler und Tester schnell eine Rückmeldung über den Erfolg oder Mißerfolg der geschriebenen Tests. Dabei ist dieses Feedback allgemeingültiger als lokal ausgeführte Tests auf dem jeweiligen Rechner des Entwicklers, weil beispielsweise nur lokale Einstellungen, die nur für einen Rechner bzw. Entwickler gelten, schnell aufgedeckt werden [QT2012].

Weiterhin wird automatisiert ermittelt welcher Prozentsatz der Funktionalität innerhalb der Software durch die Module tatsächlich abgedeckt wird. Dabei wird zunächst der Quellcode instrumentiert und dann mit den Modultests ermittelt welche Funktionen oder Statements damit getestet werden. Dies wird dann mit der insgesamt geforderten Code Coverage ins Verhältnis gesetzt.

Das Ergebnis dieses Schrittes sind also die Testergebnisse sowie die Code Coverage der Modultests (vgl. Abbildung 4).

4.3. Software Integration und Test

Ein entscheidender Teil des Software Entwicklungsprozesses bei der Entwicklung eingebetteter Systeme ist die Software Integration und der Test. Hierbei wird die entwickelte Funktionalität auf der Zielhardware integriert – meist werden dazu andere Compiler und Werkzeuge genutzt als im täglichen Entwicklungsprozess auf den Computern der Entwickler. Sehr oft ist auch mehr als eine Softwarekomponente zu integrieren. Beispielsweise wird die HMI Software für ein Fahrerinformationssystem an einem Ort entwickelt, die Systemfunktionen von einem anderem Team und/oder einer anderen Firma an einem anderen Ort und das Betriebssystem kommt von einer dritten Partei. Hier muss eine Umgebung geschaffen werden, die es ermöglicht z.B. die Anzeigesoftware, die großen optischen Veränderungen und Anpassungen unterworfen ist, auf der Zielhardware komplett zu integrieren und zu testen. Das Kompilat für die Zielhardware wird automatisiert und täglich erstellt. So erhalten die Entwickler zeitnah Rückmeldung über Warnungen und Fehler mit dem Ziel-Compiler und können diese viel schneller beheben, als wenn der Integrationsschritt immer am Schluss der Entwicklung erfolgt. Die Rückmeldung vom Buildserver erfolgt in der Regel per Email direkt an den jeweiligen Entwickler.

Als Ergebnis dieses Schritts in der Continuous Integration Kette steht eine funktionsfähige flashbare Software für die Zielhardware zur Verfügung.

5 Zusammenfassung

In diesem Artikel wird beschrieben, wie funktionale Sicherheit für ein Steuergerät aus dem Bereich der Fahrerinformation, d.h. mit grafischer Anzeige umgesetzt werden kann. Hierbei wird die Trennung der sicherheitsrelevanten und nicht sicherheitsrelevanten Software als zentrale Möglichkeit im Software Architektur Design genutzt.

Im zweiten Teil des Artikels wird ausgeführt wie für drei verschiedene Schritte in der ISO-konformen Softwareentwicklung Continuous Integration für die Entwicklung von eingebetteten Systemen genutzt wird.

Literatur

[ISO26262] ISO Norm 26262 zur Fahrzeugsicherheit.

[BS2012] Becker, Ulrich und Sechser, Bernhard, Sicherheit entwerfen statt testen., Automotive 11, 2012

[Jenkins] http://www.jenkins.org

[QT2012] Klaus Quibeldey, Christoph Thelen, Continuous Deployment, Informatik Spektrum, Springer Verlag 2012, https://www.gi.de/service/informatiklexikon/detailansicht/article/deployment-continuous.html

Kryptographische Hashfunktionen: Historie, Angriffe und aktuell sichere Standards

Christian Wenzel-Benner, Dr.-Ing. Daniel Wasserrab

Bereich Innovation Fachbereich Methoden
ITK Engineering AG
Im Speyerer Tal 6
D-76761 Rülzheim
christian.wenzel-benner@itk-engineering.de
daniel.wasserrab@itk-engineering.de

Abstract: Hashfunktionen sind das „Arbeitspferd der Kryptographie“. Sie werden in einer Vielzahl von Anwendungen und Protokollen eingesetzt und sind essentiell für die sichere Kommunikation in verteilten Systemen wie z.B. vernetzten Automobilen. Die bekannteste Anwendung von Hashfunktionen ist der Nachweis von Integrität bzw. Authentizität von Daten. Gängige Beispiele hierfür sind signierte E-Mails und Softwareupdates für Computer, Handys und Steuergeräte sowie Antivirensoftware, die Hashfunktionen auf diese Art und Weise nutzen. Darüber hinaus finden sie an vielen Stellen Verwendung, an denen Subsysteme voneinander isoliert werden müssen, z.B. bei der sicheren Schlüsselgenerierung und dem Umgang mit Passwörtern. Trotz des breiten Anwendungsspektrums wurden Hashfunktionen in der Vergangenheit nicht die notwendige Aufmerksamkeit zuteil. Dies zeigte sich 2005 im Bereich der Forschung, als ein theoretischer Angriff auf die Standard-Hashfunktion SHA-1 erfolgreich durchgeführt wurde. Die kurzfristige Reaktion war die Definition und Standardisierung der aus SHA-1 abgeleiteten Standard-Hashfunktionen SHA-256 und SHA-512. Als langfristige Lösung wurde ein Standardisierungswettbewerb für eine von Grund auf neu zu entwickelnde Standard-Hashfunktion SHA-3 ausgerufen, der im Oktober 2012 abgeschlossen wurde. Im kommerziellen Bereich war lange Zeit die veraltete und nicht offiziell standardisierte Hashfunktion MD5 sehr verbreitet. Obwohl Experten seit Mitte der 1990er Jahren vor Schwächen in MD5 warnten, wurde der Algorithmus aufgrund seines guten Laufzeitverhaltens und seiner großen Verbreitung weiterhin eingesetzt. Im Mai 2012 wurde der bisher schwerste Angriff unter Ausnutzung der Schwächen von MD5 entdeckt: das Schadprogramm „Flame“ verbreitete sich mit Hilfe einer gefälschten Microsoft-Codesignatur. Jedoch wird MD5 immer noch explizit als Beispiel für kryptographische Hashfunktionen in AUTOSAR-Security Foliensätzen erwähnt (Stand November 2012). SHA-3 ist standardisiert, jedoch erfolgt der Transfer von Wissen über Angriffe und neue Standards aus dem Bereich der kryptografischen Forschung und der IT-Sicherheit in den Bereich der industriellen Embedded-Entwicklung nicht in der Geschwindigkeit, die für die Sicherheit vernetzter Automobile wünschenswert wäre.

1 Eigenschaften, Aufbau und Charakteristika

1.1 Hashfunktionen generell

Als eine Hashfunktion bezeichnet man eine Transformationsfunktion von einem beliebigen, aber endlichen Urbild-Bereich auf einen Bild-Bereich von endlicher und geringerer Größe. Anders ausgedrückt erzeugt eine Hashfunktion zu einem beliebigen Eingabe-String einen Ausgabe-String geringerer Länge. Hashfunktionen sind im Allgemeinen nicht bijektiv, sodass schwerlich vom Bild auf das Urbild geschlossen werden kann. Da der Bildbereich von Hashfunktionen meist wesentlich kleiner ist als der Urbildbereich, sind Kollisionen, also ein gleiches Bild bei unterschiedlichen Urbildern, nicht auszuschließen. Bei zwei unterschiedlichen Bitfolgen ergibt die Hashfunktion dann die gleiche Bitfolge als Ausgabe [1, S.354]. Wichtig ist, dass eine Hashfunktion zu einer bestimmten Eingabe stets die gleiche Ausgabe erzeugt. Suchfunktionen sind eines der häufigsten Anwendungsgebiete von Hashfunktionen. Hierbei wird der Hashwert als Schlüssel zu dem Objekt, aus dem er berechnet wurde, vermerkt. Der Schlüssel hat dabei eine definierte Länge. Sucht man das Objekt O, wird zuerst der Hashwert h(O) erzeugt und anschließend alle Schlüssel der Tabelle auf Gleichheit mit h(O) überprüft. Da die Schlüssel von geringerer Größe sind als die zugehörigen Objekte, ist diese Suche sehr effizient. Für auftretende Kollisionen muss eine Behandlungsmethode gewählt werden. [2]

1.2. Kryptographische Hashfunktionen

Kollisionen bei normalen Hashfunktionen erzeugen eine erhöhte Laufzeit. Im Kapitel 1.1 wurde das Beispiel einer Hash-Tabelle angeführt. Tritt in solch einem Fall eine Kollision auf, müssen anstelle der Hashwerte die Objekte selbst verglichen werden, wodurch die Suche wesentlich verlangsamt wird. Bei kryptographischen Hashfunktionen sind solche Kollisionen noch weitaus gravierender. Hier werden die Hashwerte beispielsweise zur Verifikation von Passwörtern und zur Integritätssicherung von Nachrichten und Zuständen, die über eine unsichere Leitung versendet werden, genutzt. Bei der Integritätssicherung soll sichergestellt werden, dass die gesendete Nachricht nicht manipuliert werden kann, da sich ansonsten der Hashwert verändern würde. Ist es möglich, eine Kollision zu erzeugen, kann die übertragene Nachricht unbemerkt abgefangen und manipuliert werden. Zusammenfassend lässt sich sagen, eine kryptologische oder kryptographische Hashfunktion ist eine spezielle Form der Hashfunktion, welche kollisionsresistent und/oder eine Einwegfunktion ist [1, S.354-357]. Hierbei ist anzumerken, dass auch normale Hashfunktionen oft Einwegfunktionen sind. Es handelt sich dabei aber meist um einen Nebeneffekt und nicht um eine geforderte Eigenschaft.

1.2.1. Eigenschaften kryptographischer Hashfunktionen

Eine schwache Hashfunktion (Eckert, [1]) / Einweg-Hashfunktion (Schneier, [3]) erfüllt folgende Bedingungen:

1. Einwegfunktion: Es ist praktisch unmöglich, zu einem gegebenen Ausgabewert einen Eingabewert zu finden, den die Hashfunktion auf abbildet: (englisch: preimage resistance).

2. Schwache Kollisionsresistenz: es ist praktisch unmöglich, für einen gegebenen Wert einen davon verschiedenen zu finden, der denselben Hashwert ergibt (englisch: 2nd-preimage resistance).

3. Die Hashfunktion muss leicht zu berechnen sein. [1, S. 355]

Für eine starke Hashfunktion (Eckert) / kollisionsresistente Einweg-Hashfunktion (Schneier) gilt zusätzlich:

1. starke Kollisionsresistenz: es ist praktisch unmöglich, zwei verschiedene Eingabewerte und zu finden, die denselben Hashwert ergeben (englisch: collision resistance). Der Unterschied zur schwachen Kollisionsresistenz besteht darin, dass hier beide Eingabewerte und frei gewählt werden dürfen.

Anmerkung: es handelt sich auch bei einer starken Hashfunktionen nicht um eine injektive Funktion. Anders ausgedrückt sind auch starke Hashfunktionen nicht kollisionsfrei [1, S.354-357].

Intuitiv kommt man in die Versuchung, die Kollisionsresistenz von schwachen und starken Hashfunktionen als ähnlich stark anzusehen. Das Geburtstagsparadoxon zeigt sehr anschaulich, wie falsch diese Annahme ist. Die dem Paradoxon zugehörige Frage lautet: "Wie viele Personen müssen in einem Raum sein, damit die Wahrscheinlichkeit, dass zwei Personen am gleichen Tag Geburtstag haben, bei ca. 50% liegt." Die Antwort lautet: 23. Paradoxon wird dieses Beispiel genannt, da die meisten Menschen intuitiv eine wesentlich höhere Zahl schätzen. Legt man hingegen einen bestimmten Tag fest und fragt, wie viele Menschen haben an exakt diesem Tag Geburtstag, erreicht man erst ab einer Gruppenmenge von 183 die 50% Marke. Äquivalent ist der Unterschied bei starker und schwacher Kollisionsresistenz. Eine beliebige Kollision für einen 128bit starken Hashwert erreicht man bereits durch Berechnung von durchschnittlich 264 Werten.

1.2.2. Aufbau kryptographischer Hashfunktionen

Die meisten Hashfunktionen folgen der Merkle-Damgård-Konstruktion und sind iterierte Kompressionsfunktionen. Eine Kompressionsfunktion nimmt als Eingabe zwei Bitfolgen der Länge n und gibt eine Bitfolge der Länge m aus. Zusätzlich ist sie eine Einwegfunktion, es sollte also schwer sein, zu einer gegebenen Ausgabe passende Eingabewerte zu finden. Bei der Merkle-Damgård-Konstruktion (oft abgekürzt MD) wird die eingegebene Nachricht M zuerst in Blöcke fester Länge M1 bis Mn geteilt und mit zusätzlichen Bits aufgefüllt, so dass die Eingabelänge ein ganzzahliges Vielfaches der Blocklänge beträgt. Die Kompressionsfunktion nimmt als Eingabe die Ausgabe der

vorherigen Funktion und einen Nachrichtenblock entgegen. Bei der ersten Funktion wird ein sogenannter Initialisierungswert IV verwendet. Abbildung 1 stellt die Merkle-Damgård-Konstruktion schematisch dar.

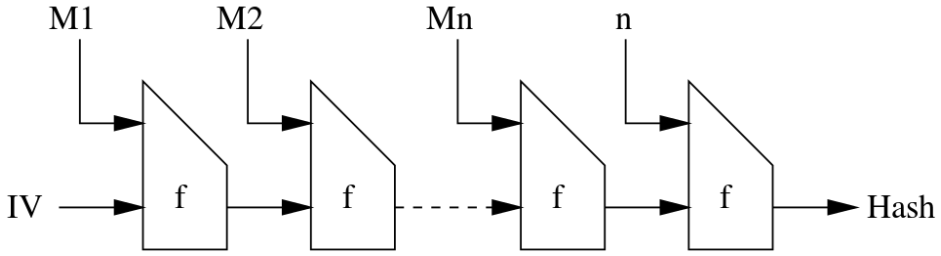


Abbildung 1: Merkle-Damgård-Konstruktion, Autor MarioS,
http://commons.wikimedia.org/wiki/File:MerkleDamgaard.svg, CC-BY-SA

Klassische MD-Konstruktionen haben einige generelle Schwächen:

1. Wird der interne Zustand (Ausgabe der Funktion f) am Ende direkt als Hashwert herausgegeben, ist eine MD-basierte Hashfunktion anfällig für Erweiterungs-Angriffe (extension attacks). Diese ermöglichen zu einer Nachricht M mit Hashwert h(M) eine Nachricht M||M‘ zu konstruieren (M‘ wird an M angehängt) und dafür den korrekten Hashwert h(M||M‘) zu berechnen, selbst wenn der öffentlichen Nachricht M noch ein geheimer Schlüssel K vorangestellt wurde [3] (MAC, siehe 2. Anwendungsgebiete).

2. Das Auffinden von Mehrfachkollisionen, bei denen mehrere Nachrichten den gleichen Hashwert erzeugen, ist nur wenig aufwändiger als das Auffinden einer einzelnen Kollision [4].

3. Mit einem selbst gewählten Hashwert h0 und einem Nachrichtenanfang M0 kann ein passendes Nachrichtenende M1 gefunden werden, sodass h0=h(M0||M1) erfüllt ist. Dies wird als Herding-Angriff bezeichnet [5].

Je nach verwendeter Kompressionsfunktion teilt man kryptographische Hashfunktionen in zwei Gruppen ein:

1. Dedizierte Hashfunktionen

Die Kompressionsfunktion von dedizierten Hashfunktionen wurde speziell für die oben genannten Eigenschaften entwickelt. Die bekanntesten Vertreter sind MD5 und die Algorithmen der SHA Familie. [1]

2. Hashfunktionen durch symmetrische Blockchiffren

Hierbei handelt es sich um Kompressionsfunktionen, die nicht explizit für die Erzeugung von Hashs entworfen wurden. Als Blockchiffre lässt sich beispielsweise AES einsetzen. Im Allgemeinen handelt es sich bei solchen Kompressionsfunktionen um schwache Hashfunktionen, da die Konstruktion von starken Hashfunktionen mit symmetrischen Blockchiffren aufwändig ist. [1]

2 Anwendungsgebiete

Eine der häufigsten Anwendungen von kryptographischen Hashfunktionen ist die Integritätsprüfung, bzw. der Schutz der Datenintegrität von Dateien und Nachrichten. Hierbei wird ein Hashwert der gesamten Bitfolge der zu schützenden Daten gebildet und an die Datei oder Nachricht angehängt. Da die ursprünglichen Daten dabei nicht verändert werden, kann der Hashwert jederzeit wieder erzeugt und mit dem angehängten Wert verglichen werden. Sind beide Hashs identisch, ist die Integrität nicht verletzt. Man nennt solche Hashs auch Manipulation Detection Codes (MDC). [6] Ähnlich verfährt man, wenn man Passwörter verschleiern möchte, um sie persistent zu speichern. Hierbei wird ein Hashwert des zu speichernden Passworts erzeugt und persistent abgelegt. Um ein eingegebenes mit den gespeicherten Passwörtern zu vergleichen, braucht man nur den Hashwert des Eingabepassworts zu bilden und mit den hinterlegten zu vergleichen. Möchte man mit einem Hash nicht nur die Integrität von Daten, sondern auch die Authentizität sicherstellen, kann man eine digitale Signatur verwenden. Hierbei wird ein Hashwert aus einer Nachricht gebildet und der Hashwert mit einem geheimen Schlüssel verschlüsselt (asymmetrische Verschlüsselung). Der Empfänger muss zur Prüfung der Signatur den Hashwert mit dem öffentlich bekannten Schlüssel des Absenders entschlüsseln und anschließend mit dem selbst berechneten Hashwert vergleichen. Da der geheime Schlüssel des Absenders nur dem Absender bekannt sein sollte, kann somit die Authentizität des Absenders als valide angesehen werden. Ein ähnlicher Anwendungsfall ist die Verwendung von Message Authentication Codes (MAC). Hierbei wird ein Schlüssel verwendet, um einen Hashwert zu erzeugen. Allerdings handelt es sich hierbei um symmetrische Verschlüsselung. Das bedeutet, Empfänger und Sender benötigen denselben Schlüssel und sind beide in der Lage, identische Hashwerte zu erzeugen. Hiermit kann nicht unterschieden werden, wer von den Geheimnisträgern den MAC angelegt und somit die Nachricht verschickt hat, aber es kann sichergestellt werden, dass es einer der Geheimnisträger gewesen sein muss. Es ist also eine schwächere Variante der Authentizität als bei digitalen Signaturen. Weitere Anwendungsfälle sind beispielsweise Pseudozufallsgeneratoren und die Konstruktion von Blockchiffren.

3. Die kryptographische Hashfunktion MD5

MD5 wurde von Ron Rivest als eine Weiterentwicklung der Hashfunktion MD4 entwickelt und im April 1992 als RFC 1321 veröffentlicht. MD5 ist eine Merkle-

Damgård-Konstruktion, die einen 128 Bit Hashwert erzeugt. Als Eingabe werden Blöcke von 512 Bit Größe genutzt, die jeweils mit dem 128 Bit Ergebnis der vorherigen Kompressionsfunktion verarbeitet werden. Für die erste Kompressionsfunktion wird ein definierter Initialisierungsvektor genutzt. Jede Kompressionsfunktion wird in vier Runden durchgeführt und dafür der Eingabeblock in sechzehn 32 Bit große Bitfolgen zerlegt. In jeder Runde erfolgen 16 Verarbeitungsschritte. Damit ergeben sich insgesamt 64 Verarbeitungsschritte pro Kompressionsfunktion. Pro Runde wird eine andere Grundfunktion und ein zyklischer Linksshift als Verarbeitungsschritt genutzt. Aufgrund des Geburtstagsparadoxon ist die Länge des Hashs von 128 Bit unzureichend, da lediglich 2^{64} Hashwerte berechnet werden müssen, um eine Kollision zu bekommen. MD5 zeigt alle Probleme von klassischen Merkle-Damgård-Konstruktionen (und noch einige mehr), einige resultierende Angriffe werden in Kapitel 5.1. Angriffe auf MD5 dargestellt.

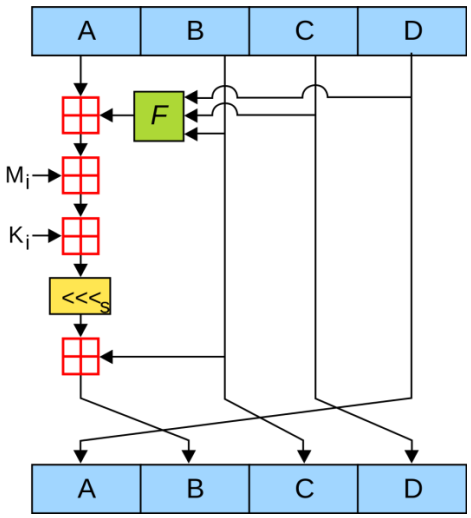


Abbildung 1: MD5 Kompressionsfunktion, Autor Surachit, http://commons.wikimedia.org/wiki/File:MD5.svg, CC-BY-SA

4. Sichere kryptographische Hashfunktionen: SHA-2 und SHA-3

Obwohl MD5 sich sehr schnell zu einem de-facto Standard entwickelte, existierte bis 1993 kein wirklicher Standard für kryptographische Hashfunktionen. Auch RFC 1321 erhebt keinen Anspruch, MD5 als Standard zu definieren: „This memo provides information for the Internet community. It does not specify an Internet standard."

4.1 SHA-0 und SHA-1

Da der Bedarf an einer standardisierten kryptographischen Hashfunktion offensichtlich war, veröffentlichte das US-Amerikanische „National Institute of Standards and Technology“ (NIST) 1993 den „U.S. Federal Information Processing Standard“ (FIPS) 180. FIPS 180 standardisiert eine kryptografische Hashfunktion namens „Secure Hash Algorithm“ (SHA). SHA ist eine Entwicklung der „National Security Agency“ (NSA) und erzeugt einen 160 Bit langen Hashwert. SHA war die erste kryptografische Hashfunktion, die als FIPS standardisiert wurde, zuvor wurde z.B. 1977 die Blockchiffre DES als FIPS 46 standardisiert. Aufgrund eines Designfehlers (der nicht detailliert erläutert wurde) zog die NSA SHA kurz nach Veröffentlichung zurück und führte eine retroaktive Umbenennung nach SHA-0 durch.

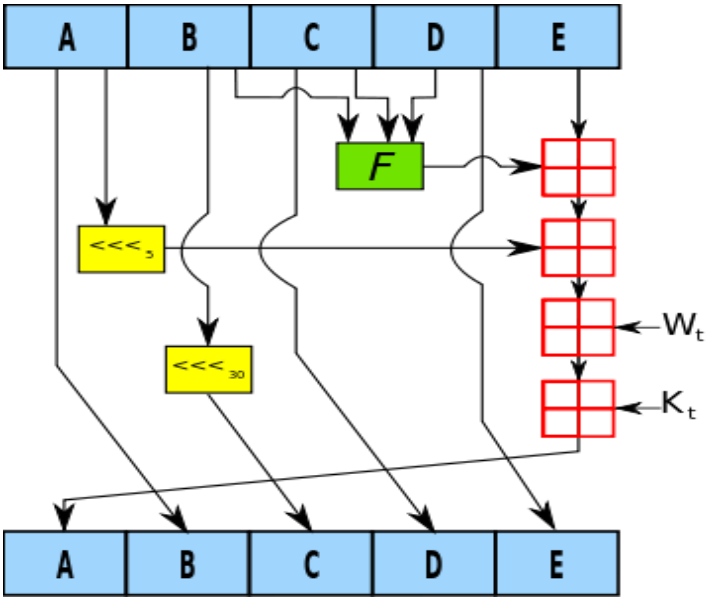


Abbildung 2: SHA-1 Kompressionsfunktion, Autor Ppietryga,
http://en.wikipedia.org/wiki/File:SHA-1.svg, CC-BY-SA

Im Zuge dessen wurde eine korrigierte Version des Algorithmus als SHA-1 veröffentlicht und 1995 mit FIPS 180-1 standardisiert. SHA-1 unterscheidet sich nur durch eine einzige zusätzliche Bitrotation in der Kompressionsfunktion von SHA-0. Abbildung 2 zeigt die Kompressionsfunktion von SHA-1. Dieser Unterschied ist jedoch signifikant, Kollisionen für SHA-0 konnten z.B. 2008 mit $2^{33.6}$ Operationen gefunden werden, wie von S. Manuel und T. Peyrin auf der Konferenz FSE präsentiert.

4.2 SHA-2

Lange vor den Angriffen von Wang et.al. wurde Bedarf an standardisierten kryptografischen Hashfunktionen mit Hashwerten länger als 160 Bit offensichtlich.

NIST reagierte mit FIPS 180-2 im Jahr 2001, welcher die ebenfalls von der NSA entwickelten Algorithmen der SHA-2 Familie standardisierte. Es gibt vier Algorithmen in dieser Familie: SHA-224, SHA-256, SHA-384, SHA-512. Die Länge der erzeugten Hashwerte in Bit ist im Namen codiert.

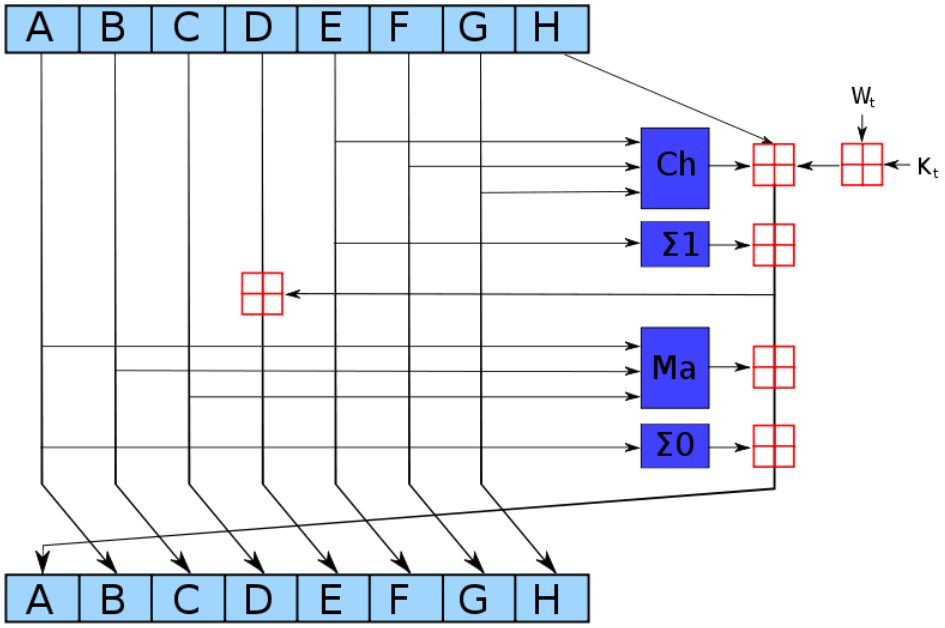


Abbildung 3 zeigt die SHA-2 Kompressionsfunktion. Die SHA-2 Familie hat sich trotz der Ähnlichkeit zu SHA-1 als robust gegenüber Varianten der Angriffe von Wang et.al. erwiesen und auch sonst gibt es bisher keine publizierten Angriffe, die effizienter sind als generische Geburtstagsangriffe (2^{112} bei SHA-224 bis 2^{256} bei SHA-512).

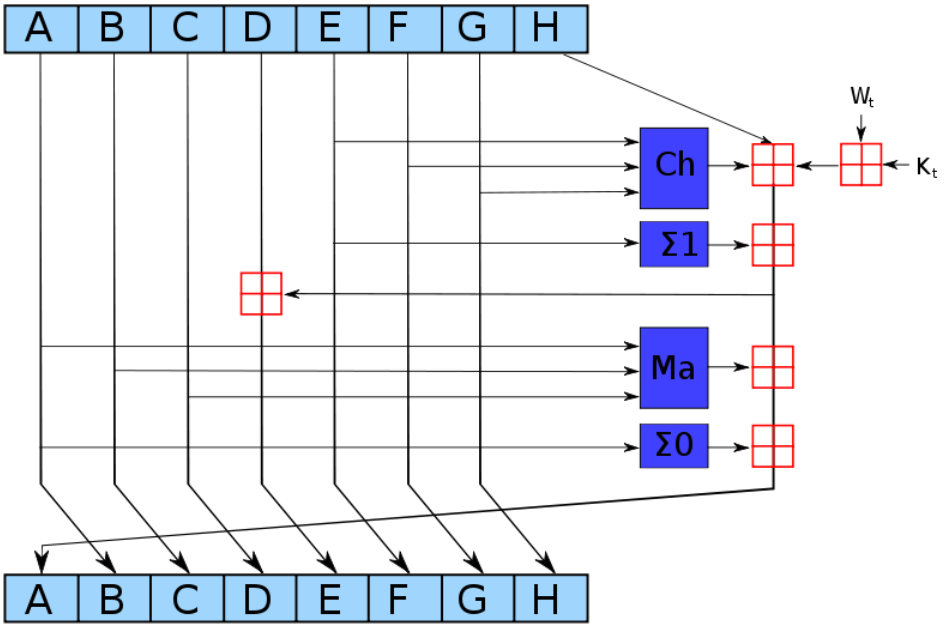


Abbildung 3: SHA-2 Kompressionsfunktion, Autor Kockmeyer
http://en.wikipedia.org/wiki/File:SHA-2.svg, CC-BY-SA

Stand Juli 2013 ist die SHA-2 Familie seitens NIST zur uneingeschränkten Verwendung durch US Regierungsbehörden freigegeben [7].

Algorithmen der SHA-2 Familie sind Merkle-Damgård-Konstruktionen und damit auch anfällig gegenüber z.B. Erweiterungs-Angriffen.

4.3 SHA-3

Im Gegensatz zu SHA-1 und SHA-2 hat NIST SHA-3 über einen weltweiten, öffentlichen Wettbewerb standardisiert. Über 80 Algorithmen wurden von Gruppen aus Forschung und Industrie sowie Privatpersonen eingereicht und in einem mehrjährigen rigorosen Ausleseprozess, an dem auch ITK Engineering beteiligt war, auf fünf Finalisten reduziert. Der Algorithmus Keccak-f[1600] hat sich unter den Finalisten durchgesetzt und den Wettbewerb gewonnen.

4.3.1 Keccak

Keccak bezeichnet eine Familie von Hashalgorithmen, die allesamt auf sogenannten „sponge“ (dt. Schwamm) Konstruktionen basieren [8]. Sponge-Konstruktionen verhalten sich ähnlich wie „normale“ Hashalgorithmen und erfüllen die Anforderungen, die unter

Fehler! Verweisquelle konnte nicht gefunden werden. definiert wurden. Aufgrund der Erfahrungen mit Merkle-Damgård-Konstruktionen konnte beim Entwurf der Sponge-Konstruktionen darauf geachtet werden, bekannte Schwächen grundsätzlich auszuschließen. Da SHA-3 so eine Sponge-Konstruktionen und keine Merkle-Damgård-Konstruktion ist, ist es, im Gegensatz zu MD5 und SHA-0,-1,-2, vollständig immun gegenüber Erweiterungs-Angriffen.

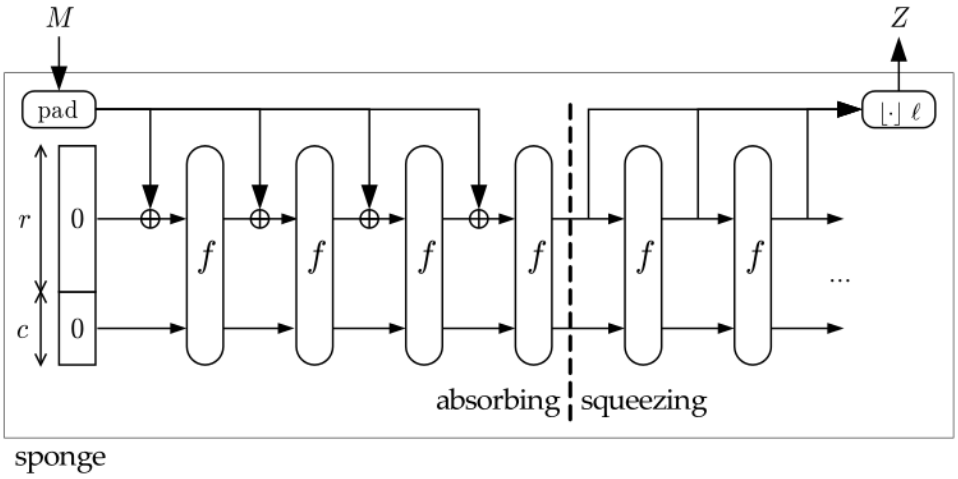


Abbildung 4: Grundprinzip Sponge, Autor: Das Keccak Team, http://keccak.noekeon.org/, CC-BY

Abbildung 4 skizziert ein Sponge-Konstrukt. Der Initialisierungsvektor, der Initial-Zustand (engl. initial-state) genannt wird, ist ein Nullvektor. Ein Teil des Initial-Zustands, der mit r bezeichnet wird, wird per XOR mit dem ersten Block der Eingabe verarbeitet. Der zweite Teil des Initial- Zustands, der mit c bezeichnet wird, wird unverändert an die Transformationsfunktion f übergeben. Die Transformationsfunktion erzeugt den nächsten Zustand. Der Zustand ist bei Keccak-f[l] eine Matrix der Größe $l=5x5x2^k$ wobei k eine Zahl zwischen 0 und 6 ist. Bei SHA-3 wird Keccak-f[1600] verwendet, womit der Zustand eine Größe von 5x5x64=1600 Bit besitzt. Die gesamte Phase der Eingabedatenverarbeitung nennt man „absorbing“. Die Phase der Ausgabeerzeugung wird als „squeezing“ bezeichnet. Hierbei wird stets ein Teil von der Größe r des Zustands zur Bildung des Hashwerts genutzt, bevor abermals die Transferfunktion ausgeführt wird [9] [10]. Da die c Bits lediglich implizit in den nächsten Zustand eingehen, ist ein Erweiterungs-Angriff (engl. extension attack) nicht möglich. Sponge-Konstruktionen können außerdem als Pseudo-Zufallszahlengenerator verwendet werden. Da die „absorbing“ und „squeezing“ Phasen sich beliebig abwechseln können, kann man eine gewisse Menge an Entropie aufnehmen und bei Bedarf ausgeben. Danach ist im Unterschied zu anderen Hashfunktionen keine neue Initialisierung nötig, sondern es ist lediglich nötig und auch möglich, neue Entropie per „absorbing“ aufzunehmen. Die tatsächliche Entropie der generierten Zufallszahlen hängt natürlich vom Grad der absorbierten Entropie ab[9]. Eine Sponge-Konstruktion lässt sich ebenfalls für authentifizierte Verschlüsselung (engl. authentificated encryption) nutzen. Hierzu wird die Transformation des Initial-Zustands zuerst mit einem definierten

Schlüssel ausgeführt. Anschließend wird der Header der zu verschlüsselnden Nachricht transformiert. Danach folgen die Blöcke des Payloads, wobei nach jedem Block ein „squeezing“ ausgeführt wird. Diese Ausgaben werden mit dem zu ver- bzw. entschlüsselnden Block des Payloads per XOR Operation verarbeitet. Der letzte Ausgabeblock wird zusätzlich als MAC für die Nachricht verwendet. Durch die gleichzeitige Verschlüsselung und Authentifizierung in einem Funktionsaufruf ist eine hoch effiziente Implementierung möglich [11].

4.3.2 NIST – Wettbewerb und Kontroverse

Die Begründung des NIST-Komitees zur Auswahl von Keccak als Sieger des SHA-3 Wettbewerbs war seine außerordentliche Flexibilität, die Möglichkeit, die Größe der generierten Hashs zu beeinflussen, oder als Verschlüsselungsalgorithmus verwendet zu werden. Als weiterer ausschlaggebender Punkt wurde die andersartige Architektur der Sponge-Konstruktion, im Vergleich zum SHA-2 Standard, aufgeführt. Weiterhin bietet Keccak eine gute allgemeine Performance und effiziente Umsetzbarkeit in Hardware-Implementierungen [12]. Das NIST-Komitee geriet jedoch in Kritik, als sie noch vor der Etablierung von SHA-3 als Standard erläuterte, dass es nur die Bitlängen 128 Bit und 256 Bit standardisieren wolle, anstatt der 224 Bit, 256 Bit, 384 Bit und 512 Bit, die in der Wettbewerbseinreichung von Keccak vorgesehen waren. Außerdem sollten weitere Änderungen an den Parametern des Algorithmus vorgenommen werden, um die Ausgabegeschwindigkeit zu erhöhen [13]. Einige Forscher kritisierten diese Anpassungen, die ihrer Meinung nach die Sicherheit beeinträchtigen und außerdem den Algorithmus so stark ändern, dass es sich nicht mehr um den untersuchten, ursprünglich eingereichten Algorithmus handeln würde. Ferner sei im Wettbewerb kein Schwerpunkt auf die Performanz der eingereichten Algorithmen gelegt worden. Das Keccak-Team reagierte auf diese Kritik an NIST und gab bekannt, dass eine Bitlänge von 128 Bit bereits eine Option in ihrem SHA-3 Vorschlag war. Jedoch empfehlen sie eine Bitlänge von 256 Bit [14]. In Bezug auf die durch NIST vorgenommenen Änderungen am Algorithmus stellte das Keccak-Team klar, dass der resultierende Algorithmus weiterhin der Keccak Familie zugehörig sei und die vorgeschlagenen Änderungen Resultat einer Diskussion des Keccak-Teams mit NIST waren [15]. Da jedoch die Bedenken in der Kryptographie-Gemeinschaft gegen die Änderungen am Algorithmus und die Beschränkung der Bitlängen so groß waren, wurden diese für das SHA-3 Draft wieder zurückgenommen [16].

4.4 RIPEMD und WHIRLPOOL

Über FIPS 180 hinaus standardisiert ISO/IEC 10118-3:2004 noch RIPEMD und WHIRLPOOL. RIPEMD ist eine Merkle-Damgård-Konstruktion mit den bekannten Schwächen (jedoch bisher nicht erfolgreich angegriffen worden), Whirlpool basiert auf Komponenten der Blockchiffre AES und ist ebenfalls bisher nicht erfolgreich angegriffen worden. ISO/IEC 10118-3:2004 wurde 2013 erneut bestätigt.

5. Angriffe auf Hashfunktionen

5.1. Angriffe auf MD5

5.1.1 Akademische Angriffe

1993 präsentierten Bert den Boer und Antoon Bosselaers einen Algorithmus [17] zum Erzeugen von Pseudokollisionen auf die Kompressionsfunktion von MD5. Dieser Angriff bedroht noch nicht die volle MD5 Funktion, weist aber Designschwächen der zentralen Komponente nach. Bruce Schneier warnt [1] bereits 1996 auf dieser Grundlage vor der Verwendung von MD5.

1996 findet Hans Dobbertin eine Kollision [18] für modifiziertes MD5 (mit geänderten Konstanten A,B,C,D). Dies verdeutlicht Designschwächen in MD5, weitere Kryptologen raten zum Umstieg.

2004 präsentieren Xiaoyun Wang und Hongbo Yu auf der EUROCRYPT MD5 Kollisionen, erzeugt mit 2^{39} Operationen [19] für zwei Nachrichten mit identischem Anfang (common/identical – prefix). Zu diesem Zeitpunkt lässt sich der Angriff auf einem gängigen IBM P690 Unix Server in ca. einer Stunde durchführen. MD5 verliert damit die schwache Kollisionsresistenz bei manchen Nachrichtenpaaren mit identischem Anfang.

2007 präsentieren Stevens, Lenstra und de Weger auf der EUROCRYPT MD5 Kollisionen mit 2^{49} Operationen [20] für zwei Nachrichten mit frei wählbarem Anfang (chosen-prefix). MD5 verliert damit die schwache Kollisionsresistenz bei einigen Nachrichtenpaaren mit vom Angreifer frei wählbarem Anfang.

2009 verbessern Stevens et.al. die Angriffe [21] mit identical-prefix auf nur noch 2^{16} Operationen und die Angriffe mit chosen-prefix auf 2^{39} Operationen.

5.1.2 Praktische Demonstrationen und Angriffe

2006 veröffentlicht Peter Selinger unter [22] die „evilize“ Bibliothek. Damit können identische MD5-Hashwerte für zwei unterschiedliche, ausführbare Programme erzeugt werden. Die Laufzeit für den Angriff beträgt einige Stunden auf Standard-PCs zum Zeitpunkt der Veröffentlichung. Dieser Angriff zerstört den Nutzen von MD5 als MDC in Virenscannern vollständig.

2007 sagen Stevens, Lenstra und de Weger erfolgreich den Ausgang der US Präsidentenwahl von 2008 voraus [23]. Als Beweis veröffentlichen sie den MD5 Hashwert eines PDF-Dokuments und legen nach der Wahl das passende PDF-Dokument vor. Das Dokument enthält die eindeutige Aussage, dass Barack Obama die Wahlen gewinnen wird. Da eine kryptographische Hashfunktion nicht umkehrbar und kollisionsresistent ist, „beweist“ das die Fähigkeit des Teams, in die Zukunft zu blicken.

In Wahrheit existieren zwölf PDF Dokumente mit identischen MD5 Hashwerten. Die Aktion ist also in Wirklichkeit ein Beweis dafür, dass MD5 als kryptografische Hashfunktion nicht mehr geeignet ist.

2009 demonstrieren Stevens et.al. neben den reinen prefix-Angriffen auch eine praktische Anwendung [21] namens Bösartige Zertifizierungsstelle (rogue CA), indem sie ganz offiziell ein Webseiten-Zertifikat kaufen, das unter Benutzung von MD5 signiert ist, und mittels MD5 Kollision ein zweites Zertifikat erstellen, das denselben MD5-Hashwert hat wie das gekaufte, offizielle Zertifikat. Das selbst erstellte Zertifikat ist aber nicht für eine Webseite, sondern für eine Zertifizierungsstelle und kann daher nun beliebige Webseitenzertifikate unterschreiben. Damit können Stevens et.al. Webseiten einrichten, die sich als beliebige Bank, Online-Kaufhaus oder auch Regierungsbehörde ausgeben können. Aufgrund der „korrekten" Zertifikatskette zeigen Webbrowser auf Nachfrage dem Benutzer „Beweise" dafür, dass diese Webseiten echt und die übertragenen Daten sicher sind.

2012 wird das Schadprogramm „Flame", welches bereits seit 2010 (nach manchen Quellen seit 2007) aktiv war, entdeckt. Flame ist der Super-GAU für digital signierte Software, der Schädling bringt einen kryptographischen „Beweis" dafür mit, dass er ein legitimes Microsoft Windows Update ist [24]. Computer, die in einem lokalen Netz mit einem befallen System sind, installieren Flame automatisch und ohne Benutzereingriff im Zuge ihrer regulären, aus IT-Sicherheitssicht absolut notwendigen, Suche nach Windows Updates. Durch die Schwächen von MD5 wird jedes Windows 7 System auf der Welt kompromittierbar.

5.2 Angriffe auf SHA-1

Der beste aktuell bekannte Angriff auf SHA-1 benötigt 2^{61} Operationen gemäß Marc Stevens Vortrag auf der SHARCS Konferenz 2012. Die aktuellen Angriffe auf SHA-1 basieren auf der Arbeit von Xiaoyun Wang, Yiqun Lisa Yin, und Hongbo Yu, welche 2005 einen Angriff auf SHA-1 mit 2^{69} Operationen veröffentlichten. Zu diesem Zeitpunkt war nicht absehbar, wie schnell dieser Angriff verbessert werden würde und ob er auf die 2001 mit FIPS 180-2 standardisierten Algorithmen der SHA-2 Familie übertragbar sein könnte. Da die SHA-2 Familie einige strukturelle Ähnlichkeiten zu SHA-1 aufweist, verlangten viele Kryptologen, z.B. Bruce Schneier [25], nach einem neuen, strukturell radikal anderen Hashstandard.

NIST adressierte diese Probleme mit der Abkündigung der aktiven SHA-1 Nutzung für US Regierungsbehörden ab 2010 [26], der Empfehlung für Anwender schnellstmöglich auf SHA-2 zu wechseln und der Organisation eines weltweiten, öffentlichen Wettbewerbs zur Standardisierung einer neuen kryptografischen Hashfunktion SHA-3 [27].

SHA-0 und SHA-1 sind Merkle-Damgård-Konstruktionen und damit auch anfällig gegenüber z.B. Erweiterungs-Angriffen.

Fazit

Der nicht standardisierte Algorithmus MD5 ist für angriffssichere Implementierungen nachweislich nicht mehr geeignet und sollte daher generell nicht mehr verwendet werden. Obwohl es Anwendungsszenarien gibt, für die noch kein erfolgreicher Angriff auf MD5 publiziert wurde, warnen Niels Ferguson und Bruce Schneier generell davor, Algorithmen einzusetzen, die nur in manchen Szenarien sicher sind [28], Kap 5.4.

Für langfristig sichere Anwendungen wird SHA-1 inzwischen nicht mehr empfohlen.

Die SHA-2 Familie und SHA-3 sind uneingeschränkt nutzbar und in FIPS 180 standardisiert.

RIPEMD und Whirlpool sind nach heutigem Kenntnisstand sicher, aber nicht so modern wie SHA-3 und nicht so verbreitet (und damit intensiv analysiert) wie SHA-2.

Literaturverzeichnis

[1] Eckert, C.: IT-Sicherheit, 4. Auflage. München: Oldenbourg Verlag 2006

[2] Knuth, D.: The Art of Computer Programming 3. Sorting and Searching,

2. Ausgabe. Amsterdam: Addison-Wesley Longman 1998

[3] Schneier, B.: Angewandte Kryptographie. München: Addison-Wesley 1996

[4] Joux, A.: Multicollisions in iterated hash functions. Application to cascaded construction. Lecture Notes in Computer Science, Vol. 3152. Heidelberg: Springer- Verlag, 2004

[5] Kelsey, J. und Kohno, T.: Herding Hash Functions and the Nostradamus Attack. Lecture Notes in Computer Science, Vol. 4004. Heidelberg: Springer-Verlag, 2006

[6] Preneel,B.: Analysis and Design of Cryptographic Hash Functions. PhD Thesis, Leuven, 1993. http://homes.esat.kuleuven.be/~preneel/phd_preneel_feb1993.pdf

[7] NIST: Current policy regarding hash functions. http://csrc.nist.gov/groups/ST/hash/policy.html

[8] Keccak team: Keccak specification. http://keccak.noekeon.org/specs_summary.html

[9] Keccak team: The Sponge Functions Corner. http://sponge.noekeon.org/

[10] Keccak team: Inside Keccak. http://www.ulb.ac.be/di/scsi/sha3/resources/Gilles-Van- Ascche---Inside-Keccak.pdf

[11] Keccak team: Duplexing the sponge. http://eprint.iacr.org/2011/499.pdf

[12] NIST: Third Round Report of the SHA-3 Cryptographic Hash Algorithm Competition. http://nvlpubs.nist.gov/nistpubs/ir/2012/NIST.IR.7896.pdf

[13] John Kelsey: SHA3, Past, Present, and Future. Invited Talk. CHES 2013.

[14] Keccak team: A concrete proposal. http://keccak.noekeon.org/a_concrete_proposal.html

[15] Keccak team: Yes, this is Keccak! http://keccak.noekeon.org/yes_this_is_keccak.html

[16] NIST Computer Security Division (CSD): SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions. NIST.

http://csrc.nist.gov/publications/drafts/fips-202/fips_202_draft.pdf

[17] den Boer, B. und Bosselaers, A: Collisions for the compression function of MD5. Lecture Notes in Computer Science, Vol. 765. Heidelberg: Springer-Verlag, 1994

[18] The Status of MD5 After a Recent Attack, RSA Laboratories CryptoBytes, RSA Data Security, Bedford 1996

[19] Wang, X und Yu, H.: How to Break MD5 and Other Hash Functions. Lecture Notes in Computer Science, Vol. 3494. Heidelberg: Springer-Verlag, 2005

[20] Stevens, M., Lenstra A.K. und de Weger, B.: Chosen-Prefix Collisions for MD5 and Colliding X.509 Certicates for Different Identities. Lecture Notes in Computer Science, Vol. 4515, Heidelberg: Springer-Verlag, 2007

[21] Stevens, M., Lenstra A.K., de Weger, B et.al.: Short Chosen-Prefix Collisions for MD5 and the Creation of a Rogue CA Certificate. Lecture Notes in Computer Science, Vol. 5677, Heidelberg: Springer-Verlag, 2009

[22] Selinger, P.: MD5 Collision Demo. http://www.mathstat.dal.ca/~selinger/md5collision/

[23] Stevens, M., Lenstra A.K. und de Weger, B.: Predicting the winner of the 2008 US Presidential Elections using a Sony PlayStation 3. http://www.win.tue.nl/hashclash/Nostradamus/

[24] Heise Security: Windows Update kompromittiert

http://www.heise.de/security/meldung/Windows-Update-kompromittiert-1605393.html/

[25] Schneier, B.: Essay 74. http://www.schneier.com/essay-074.html

[26] NIST: Policy regarding hash functions as of 2006. http://csrc.nist.gov/groups/ST/hash/policy_2006.html

[27] NIST: Comments on cryptanalytic attacks on SHA-1(2006) http://csrc.nist.gov/groups/ST/hash/statement.html

[28] Ferguson, N., Schneier, B., Kohno, T.: Cryptography Engineering . Indianapolis: Wiley 2010

Implementation and Adaptation of the Pseudonymous PKI for Ubiquitous Computing for Car-2-Car Communication

Stefan Kaufmann

IT-Designers Gruppe
STZ-Softwaretechnik
Im Entennest 2
73730 Esslingen
stefan.kaufmann@stz-softwaretechnik.de

Abstract: Car-2-Car communication requires the use of pseudonymous public keys to ensure the vehicle's privacy by making its messages unlinkable over a longer period. The Pseudonymous PKI (PPKI) for Ubiquitous Computing by K. Zeng [Ze06] offers peer-side pseudonym generation, non-repudiation and an efficient revocation mechanism that excludes the revoked peers from the group of authorized participants. In addition, this work provides a new authorisation scheme that incorporates efficiently into the pseudonymous PKI, offers peer-side pseudonymous credential generation and preserves the peer's privacy during the initial authorisation obtainment from third party authentication authorities. The PKI scheme is also supplemented with a particular non-pseudonymous variation for roadside units as well as other adaptations based on Car-2-Car requirements. Finally, the signature verification performance and message sizes are analysed. We argue that the PPKI fulfils the security challenges of Car-2-Car communication and has advantages over a classical PKI.

1 Introduction

Vehicular ad-hoc networks (VANETs), also known as Car-2-Car communication, improve driver safety. Examples of new safety applications enabled by this technology are the timely detection of a traffic jam, collision warnings of obscured vehicles or the warnings about hazards on the road. Once a sufficient saturation of Car-2-Car-enabled vehicles has been reached, road users can automatically warn each other and consequently prevent various accidents. One of the main requirements for a successful rollout of this technology is trustworthiness. Otherwise, if safety messages are not trustworthy, consumers will never adopt this new technology. Beside well-functioning sensors, communication devices and applications, security is a crucial requirement. The main security challenges for the exchange of Car-2-Car messages are authenticity, integrity of messages as well as non-repudiation and privacy.

The following states at first the security requirements for Car-2-Car Communication, the current solution. Afterwards the PPKI Scheme and its adaptations, including a new

authorisation scheme, is explained. Finally, a signature sizes and performance are compared with the current solution.

1.1 Classical Public Key Infrastructures

The present solution, to provide message security, is to implement a classical Public Key Infrastructure (PKI). A certificate authority (*CA*) signs the public keys of the peers, who then digitally sign their messages. A classical PKI solves all security challenges of Car-2-Car communication apart from privacy. To achieve privacy, it is important to prevent the possibility to link messages and peers over a longer time.

1.2 Pseudonymous Public Keys

With a classical PKI, each peer uses its private key to create message signatures. The CA-signed public key will be additionally attached to this signature, which allows other peers to perform a validation. However, the peer's public key represents the main link between all of its messages and, therefore, can be seen as a unique identifier.

The obvious solution is repeatingly to change the public key (and all other identifying attributes). Total anonymity, however, is not desirable, since misbehaving nodes need to be identified and potentially removed from the group of authorized peers (revocation). Such keys are called Pseudonymous Public Keys (PPKs). They fulfil the challenge of non-repudiation.

1.3 Car-2-Car Reference Architecture

The ETSI specifications 102 940 [ETSI1] and 102 941 [ETSI2] define an ITS security reference model, consisting of different types of authorities and peers. The *Enrolment Authority (EA)* is responsible for enabling peers to generally take part in the Car-2-Car communication. The *EA* will be referred to as *CA* within this document. The *Authorisation Authority (AA)* is responsible for authorising a peer "to use of particular application, service, or privilege". For example, the use of specific Car-2-Car safety messages should be subject to those privileges. A *Manufacturer* is an authority that issues each peer a canonical identifier. The *Root CA* issues certificates to all authorities that should take part in the Car-2-Car communication.

1.4 Current Solution

A classical PKI implementation was chosen for the current Pilot PKI of the European Car-2-Car Communication Consortium [ES12]. A vehicle in this infrastructure should receive different types of certificates:

- A long-term certificate, which is only used to authenticate against other authorities.

- A specific amount of short-term certificates which are used as pseudonyms for authentic inter-vehicular communication. Those are issued from so-called *Pseudonym CAs*. Each one is only valid for a given period.
- Authorisation Certificates (Credentials) from the *AA*. They are not available in the current Pilot PKI. Since a credential "should be an unlinkable proof of authentication to the canonical identifier", an obvious solution is, that the peer transfers all of its pseudonymous public keys to the *AA*, which assigns the credentials to each of them.

When all pseudonymous public keys or authorisation credentials have expired, the peer needs to interact again with the *Pseudonym CA* and *AA* to receive another set of keys and credentials.

Peer revocation requires the revocation of the peer's long-term certificate and the prevention of the use of the pseudonymous keys, which can be achieved in two different ways: Firstly, certificate revocation lists (CRLs) can be used, which need to include all the peer's non-expired pseudonymous public keys. The second option is to specify a pseudonymous key update rate so that the peer runs out of certificates very soon.

1.5 Zeng's Pseudonymous PKI (PPKI)

The Pseudonymous PKI for Ubiquitous Computing [Ze06] is a universal scheme for authentic communication with the focus on both effective privacy and traceability in case of misbehaviour. The scheme uses pairing-based cryptography to offer the following features: It enables peer nodes to generate signed pseudonymous public keys (PPKs) themselves, which can only be linked to a peer's identity by the *CA*. The signature of the PPKs can be validated using the global group public key (GPK). Revocation is accomplished by the recalculation of the GPK, which then prohibits the revoked peer to sign new pseudonymous keys. Since different versions of the PPKs cannot validate each other, the peer with the lower version is forced to update its revocation list. The revocation list is then used to calculate the keys according to the new version.

The following explains the cryptographic foundations of pairings, states the PPKI protocols and examines if and how the PPKI can be adapted to Car-to-Car Communication. Also, analysis of the signature sizes and validation performance will be illustrated.

1.6 Pairing-Based Cryptography

Pairing-based cryptography is a rather new cryptosystem. The first pairing-based schemes were developed around the year 2000 [BF00]. The most general form [Ly07] of a pairing is a bilinear map e used for cryptographic applications is the following:

$$e: \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathcal{G}$$

where $\mathbb{G}_1$, $\mathbb{G}_2$ and $\mathcal{G}$ are groups of prime order p. It requires the following properties (using $g_1 \in \mathbb{G}_1, g_2 \in \mathbb{G}_2$) :

- bilinearity, such that $e({g_1}^a, {g_2}^b) = e(g_1, g_2)^{ab}$ for all $a, b \in \mathbb{Z}$
- nondegeneracy, such that $e(g_1, g_2) = 1$ for all g_1 if and only if $g_2 = 1$ or for all for all g_2 if and only if $g_1 = 1$

Using $g_1 \in \mathbb{G}_1, g_2 \in \mathbb{G}_2$, $g \in \mathcal{G}$ and $(a, b, x, y, z) \in \mathbb{Z}_p$, following main cryptographic hard problem-assumptions have been found in bilinear maps [Ly07]:

- Co-Computational Diffie-Hellman (co-CDH) Problem:
  Given $g_1, g_2, {g_2}^x$, compute ${g_1}^x$

- External Diffie-Hellman (XDH) Problem:
  Given $g_1, g_2, g, {g_1}^x, {g_2}^y \; and \; g^z$, decide whether xy = z

- Q-Strong Diffie-Hellman (q-SDH) Problem:
  Given $(g_1, [g_2, {g_2}^x, {g_2}^{2x}, \dots, {g_2}^{qx}])$, find $c, {g_1}^{1/(x+c)}$ for any $c \in \mathbb{Z}_p$

In addition, following theorems are relevant for the PPKI [Ze06]:

- BB Theorem based on the q-SDH Problem:
  Given $g_1, g_2, A = {g_2}^a$, find (t, x) such that $e(t, A \cdot {g_2}^x) = e(g_1, g_2)$

- Zeng's General Theorem based on the BB Theorem:
  Given $g_1, g_2, A = {g_2}^a$ and $(h_1, h_2, \dots, h_k) \in \mathbb{G}_2$, find (t_j, x) such that $\left(t_j, A \cdot {g_2}^x\right) = e(h_j, g_2)$

Bilinear maps are based on elliptic curve cryptography (ECC). The Groups $\mathbb{G}_1$ and $\mathbb{G}_2$ in the pairing e: $\mathbb{G}_1 \times \mathbb{G}_2 \to \mathcal{G}$ are subgroups of points on an elliptic curve over a field and $\mathcal{G}$ is a subgroup of a multiplicative group of a finite Field [GPS06].

For the implementation of a pairing, several algorithms exist. The first discovered ones were the Weil and the Tate pairing [Ly07]; however, faster pairing algorithms were developed [LLP08], [Ve10]. Today, the Optimal Ate pairing is one of the most efficent solutions [BG10].

2 The PPKI Scheme

The following scheme was defined in a universal form [Ze06]. It will be explained shortened and with slight modifications where the PPKI scheme was too generic for an implementation. In particular, the PPKI scheme does not require a specific signature algorithm; instead it uses a generic notation for a zero-knowledge proof of knowledge.

The present implementation (see below in section Implementation) uses the Schnorr scheme [Sc89], since Zeng already illustrated its use. In addition, through the utilisation of a self-feedback mode, the scheme makes this implementation very secure [CM09]. However, other signature algorithms can be chosen as well.

CA Initialisation

The *CA* will be initialised using the following procedure: **1.** Generate a random private key $a \in_R \mathbb{Z}_p$. **2.** Compute the public key $A = {g_2}^a \in \mathbb{G}_2$. **3.** Generate the random elements $h_1, h \in_R \mathbb{G}_2$. **4.** Publish the public key $PK_{Ver} = (p, \mathbb{G}_1, \mathbb{G}_2, \mathcal{G}, g_1, g_2, e, h_1, h, A)$ and the current version Ver, which starts at 1.

Peer Initialisation

The peer will be initialised using the following procedure: **1.** Generate a random private-key $x \in_R \mathbb{Z}_p$. **2.** Calculate the root public key $y = h^x$. **3.** Create a peer identifier ID. This should be provided ex-factory and the peer must not be able to change it afterwards. In addition, a proof s_{ID} for the peer's ownership of ID and public key must be generated. This will be discussed later in the Car-2-Car adaptation.

Peer Registration

After the initialisation the peer can register with the CA: **1.** Peer computes $y' = {g_2}^x$ and sends (ID, y, y') to the *CA*. **2.** *CA* verifies that $e(y, g_2) = e(h, y')$ holds and that the proof s_{id} is correct. **3.** *CA* generates a salt $\xi \in_R \mathbb{Z}_p$ and computes $z = Hash(ID|y|s_{id}|\xi)$ (| denotes concatenation). **4.** *CA* computes $\left(t_g = {g_1}^{1/(a+z)}, t_h = (h_1 \cdot h^x)^{1/(a+z)}\right)$. **5.** *CA* stores (ID, y, s_{id}, y', ξ) in a database, to enable the later tracing of PPKs. **6.** *CA* sends certificate $\left(t_g, t_h, z\right)$ to the peer. **7.** Peer verifies the certificate by ensuring that $e\left(t_g, A \cdot {g_2}^z\right) = e(g_1, g_2)$ and $e(t_h, A \cdot {g_2}^z) = e(h_1 \cdot h^x, h_2)$ hold. **8.** Peer computes $v_1 = e(g_1 \cdot h_1, g_2) \in \mathcal{G}$, $v_2 = e\left(t_g \cdot t_h, {g_2}^{-1}\right) \in \mathcal{G}$ and $v_3 = e(h, g_2) \in \mathcal{G}$. Since those terms are required quite often, the storage of their results (v_1, v_2, v_3) brings some performance gain. **9.** Finally, the peer stores the CA-certificate $\left(t_g, t_h, z\right)$ and the accelerators (v_1, v_2, v_3) in accordance with the current version Ver.

Pseudonymous Public Key Generation and Verification

When required, a peer can generate a new pseudonymous public key (PPK) using the following procedure: **1.** Chose a random integer $r \in_R \mathbb{Z}_p$. **2.** Compute the PPK $\left(t = \left(t_g \cdot t_h\right)^r, t_y = t^x\right)$. **3.** Generate the PPK Signature PPK_{sig} using the Schnorr Scheme: Firstly, chose three random Elements $x_1 \in_R \mathbb{Z}_p$, $x_2 \in_R \mathbb{Z}_p$ and $x_3 \in_R \mathbb{Z}_p$. Secondly, compute $R_A = {v_1}^{x_1} \cdot {v_1}^{r \cdot x_2} \cdot {v_3}^{x_3} \in \mathcal{G}$, $R_X = t^{-x_3} \cdot {t_y}^{x_1} \in \mathbb{G}_1$ and the Hash $c_s = Hash(R_A|R_X|Ver) \in \mathbb{Z}_p$. Thirdly, compute the three exponents $s_1 = x_1 - c_s \cdot r \in \mathbb{Z}_p$, $s_2 = x_2 - c_s \cdot z \in \mathbb{Z}_p$ and $s_3 = x_3 - c_s \cdot r \cdot x \in \mathbb{Z}_p$. Those four Elements represent the signature: $PPK_{sig} = (c_s, s_1, s_2, s_3)$. **4.** A verifying peer can validate the PPK_{sig} by computing ${R'}_A = {v_1}^{s_1} \cdot e(t, {g_2}^{-s_2} \cdot A^{c_s}) \cdot {v_3}^{s_3} \in \mathcal{G}$, ${R'}_X = t^{-s_3} \cdot {t_y}^{s_1} \in \mathbb{G}_1$, as well as ${c'}_s = ({R'}_A|{R'}_X|Ver) \in \mathbb{Z}_p$ and by verifying that ${c'}_s = c_s$.

The random integer $r \in_R \mathbb{Z}_p$ allows the generation of p different PPKs. The verification of a PPK_{sig} requires three operations in $\mathcal{G}$, which represents the bottleneck in this protocol, as shown later.

Message Signature Generation and Verification

The previously generated PPK will be used to sign a message. This message has to contain a timestamp, which is provided through the GeoNetworking protocol [ETSI3]. The proving peer has to perform the following actions: **1.** Compute the signature for the message m: Firstly, generate a random element $x_{1m} \in_R \mathbb{Z}_p$. Secondly, compute the message hash c_{sm}: $R_m = t^{x_1} \in \mathbb{G}_1$ $c_{sm} = Hash(R_m, m) \in \mathbb{Z}_p$. Thirdly, compute $s_m = x_{1m} - c_{sm} \cdot x \in \mathbb{Z}_p$. Finally, those two elements form the signature $S_M = (c_{sm}, s_m)$. **2.** A verifying peer can validate the message by computing c'_{sm}: $R'_m = t^{s_m} \cdot {t_y}^{c_{sm}} \in \mathbb{G}_1$, as well as $c'_{sm} = Hash(R'_m, m) \in \mathbb{Z}_p$ and by verifying that $c'_{sm} = c_{sm}$.

Tracing

When a misbehaving peer needs to be identified by linking one of its PPKs to its ID, the CA has to perform the following steps: **1.** Iterate through all database entries, which were created during the peer-registration process and check each y'_i if $e(t, y'_i) = e(t_y, g_2)$ using the $PPK(t, t_y)$. When a match is found, then all corresponding information (ID, y, s_{id}, y', ξ) is also discovered. **2.** The tracing result now can be signed and published by the *CA*. However, it is important to keep the element y' back, since this can be used to find all $PPKs$ of this peer, which would constitute a privacy breach.

Peer Revocation

When a peer was identified, it is possible to ban it from any further Car-2-Car communication. Therefore, the *CA* has to create a new version of its public key: **1.** Recompute the hash $\hat{z} = Hash(\widehat{ID}|\hat{y}|\widehat{s_{id}}|\hat{\xi})$ using the database entry of the identified peer. **2.** Compute its new public key $\tilde{g}_1 = {g_1}^{1/(a+\hat{z})}$ $\tilde{g}_2 = {g_2}^{1/(a+\hat{z})}$ $\tilde{A} = g_2 \cdot {\tilde{g}_2}^{-\hat{z}}$. **3.** Increment Ver and publish the new $PK_{Ver} = (p, \mathbb{G}_1, \mathbb{G}_2, \mathcal{G}, \tilde{g}_1, \tilde{g}_2, e, h_1, h, \tilde{A})$. **4.** Publish the revocation data $(Ver, \tilde{g}_1, \tilde{g}_2, \hat{z})$

Since communication between peers using different versions of the public key is impossible, each peer needs to update its copy of the CA's public key as well as its CA-certificate using the revocation data: **1.** Compute $\tilde{A} = g_2 \cdot {\tilde{g}_2}^{-\hat{z}} \in \mathbb{G}_2$. **2.** Generate the new part of its CA-certificate $t_g = (\tilde{g}_1 / t_g)^{1/(z-\hat{z})} \in \mathbb{G}_1$. **3.** Verify the validity of the revocation data using the same procedure as during the registration process, by checking if $e(t_h, A \cdot {g_2}^z) = e(h_1 \cdot h^x, g_2)$ holds. **4.** Store the new $PK_{ver} = (p, \mathbb{G}_1, \mathbb{G}_2, \mathcal{G}, \tilde{g}_1, \tilde{g}_2, e, h_1, h, \tilde{A})$ and the revocation data. **6.** Update the accelerators $v_1 = e(\tilde{g}_1 \cdot h_1, \tilde{g}_2) \in \mathcal{G}$, $v_2 = e(\tilde{t}_g \cdot t_h, {\tilde{g}_2}^{-1}) \in \mathcal{G}$ and $v_3 = e(h, \tilde{g}_2) \in \mathcal{G}$. **7.** Store the new CA-certificate (t_g, t_h, z) and the updated accelerators (v_1, v_2, v_3) in accordance to Ver. It can be seen that a revoked peer is unable to update the certificate, since $z = \hat{z}$ and consequently a division by zero would occur. Now this peer is unable to take part in the communication as a sender or verifying peer.

3 PPKI Adaptation for Car-2-Car Communication

The following explains the design of a hierarchical PKI architecture using the PPKI, the integration of an AA, which issues pseudonymous authorisation certificates and an efficient non-pseudonymous solution for roadside units (RSUs).

3.1 Hierarchical PKI

As already suggested by other PKI schemes for VANETs [Pa08], the high number of countries in Europe results in a substantial regulatory and administrative effort, which makes it necessary to divide *CAs* into regional authorities. If it will be possible to unite all European countries under one single *CA*, then this concept might be unnecessary. However, regional domains also imply regional and therefore smaller certificate revocation lists.

As previously demanded by the ETSI specification, a *Root CA* is required to establish trust between different regional *CA* and *AA* authorities. The *Root CA* uses a standard ECC-based signature scheme to sign the public keys of the authorities. The creation of signature s for the message m using the signee's private key x and the known generator g will be stated as $s = sign_{g,x}(m)$. An exemplary scheme is the Schnorr Signature [Sc89], which will be used here in the following form: **1.** Generate a random element $k \in_R \mathbb{Z}_p$. **2.** Compute $R_s = g^k \in \mathcal{G}$ and the hash $c_s = Hash(R_s|m) \in \mathbb{Z}_p$. **3.** Compute the element $s = x \cdot c_s + k \in \mathbb{Z}_p$ using private key x. **4.** A verifying peer or authority can validate s by computing $R_s = {g_1}^s \cdot (g^x)^{-c_s} \in \mathcal{G}$, as well as $c'_s = Hash(R_s|m) \in \mathbb{Z}_p$ and by verifying that $c'_s = c_s$.

To reduce the signature length, a hierarchical PKI architecture without chaining is chosen for the design. This requires the *Root CA* to sign the public keys of all involved authorities: $rootCert_{A_i} = sign_{g,r}(A_i)$. Also, the *Root CA* specifies the ECC parameters and the generator g, which are used for all non-pseudonymous signatures (e.g. manufacturer signatures).

To enable peers to register at any regional *CA* it is important to provide a standardised credential that proves them as valid Car-2-Car peers. According to the Car-2-Car reference architecture, a Manufacturer Authority sets the peer's canonical identifier ID. This will be enhanced with the issuance of a proof of ownership of the public key and the ID. The manufacturer creates $s_{ID,i} = \{sign_{g,m}(y_i|ID_i), M = g^m, rootCert_M\}$ for the peer's ID and public key. The peer has a private and public key dedicated to the *Root CA*. This set is later used to sign and verif the peer's public keys belonging to a regional CA, which is called the regional public key signature. A requirement for this signatures is, that the ECC parameters and the generator g are valid for the entire PKI. Having this s_{id}, the CA can validate the peer and then issue a certificate for pseudonymous communication within the CA's managed region. A peer only needs to register once at a CA. It then stores the necessary information to easily switch the keys when passing different administrative

regions. In case of a revocation, the s_{ID} needs to be sent to all CAs where the peer will be either locally revoked or excluded from future registrations.

3.2 Authorisation and Credentials

Based on the fundamental work of Verheul [Ve01], the following new pseudonymous credential scheme could be integrated into the PPKI: To retain the anonymity of the peer, the *AA* issues a blind signature for the requester's CA public key. An AA is always assigned to one CA, since it builds upon the CA's public key, to enable efficient pseudonymous certificate proofs by the peer. On the other hand, this means for a peer that changes to another a regional *CA*, that it also has to interact with a corresponding AA to generate new certificates. The credential information $CRED$ contains the Credential String CS, which denotes the specific authorisation and optionally an expiration date, the blinded signature s_{cb}, and the AA's certified public key. A message with one credential now contains the elements:

$$\text{M}=(content, S_M, PPK, PPK_{sig}, CRED)$$

The AA uses its local CA's public key and is set up the following way: **1.** Generate private key $b \in_R \mathbb{Z}_p$ and the public key $B = g_2^b$ **2.** Interact with the $root\ CA$ to get a signature for B: s_B.

A peer can acquire a credential and its signature from an AA: **1.** Peer performs a self-blinding on its CA public key $\check{y} = y^f$, $\check{h}_1 = h_1^f$, $\check{y}' = g_2^{fx}$ and $\check{h}' = g_2^f$ using $f \in_R \mathbb{Z}_p$. **2.** Peer sends to the AA the following: $\check{y}, \check{h}_1, \check{y}', \check{h}'$, its current PPK and PPK_{sig}, as well as the requested credential information CS and a proof of the right of obtainment of the requested credential s_c. The proof s_c can, for instance, be bound to this particular PPK. **3.** AA checks the peer's request, by verifying the PPK using PPK_{sig}, the proof s_c and the blinded public key by checking that $e(t, \check{y}') = e(t_y, \check{h}')$, $e(\check{h}_1, \check{y}') = e(\check{y}, \check{h}')$ and $e(h_1, \check{h}') = e(\check{h}_1, g_2)$ hold. **4.** AA then creates the blind signature using $c_r = hash(CS)$, as well as $\check{s}_c = (\check{h}_1 \cdot \check{y})^{1/(b+c_r)}$ and returns (CS, $\check{s}_c, B, s_B$) to the peer. **5.** Peer obtains the real signature $s_c = \check{s}_c{}^{1/f}$ and stores (CS, s_c, B, s_B).

The PPK and PPK_{sig} generation process will be enhanced with following steps: **1.** Peer performs self-blinding to its certificate signature using the same random integer r as used for the PPK creation: $s_{cb} = s_c{}^r$. The pseudonymous credential is composed of $CRED = (CS, s_{cb}, B, s_B)$. **2.** Peer adds an additional proof of the validity of s_{cb} to the PPK_{sig}: Firstly, calculate $R_c = v_4{}^{x_1} \cdot v_3{}^{x_3} \in \mathcal{G}$ by using the new accelerator $v_4 = (h_1, g_2)$ and use it as additional information for the hash: $c_s = Hash(R_A | R_X | Ver | R_c) \in \mathbb{Z}_p$. The new signature $PPK_{sig} = (c_s, s_1, s_2, s_3)$ has still the same amount of elements, yet is also used as a link between s_{cb}, B and the $c_r = Hash(CS)$. **2.** A verifying peer has to perform the following additional steps during the validation: Firstly, Check if B is already a trusted AA public key and if not, validate B using s_B. Secondly, generate $c_r = hash(CS) \in \mathbb{Z}_p$. Thirdly, compute $R'_c = v_4{}^{s_1} \cdot v_3{}^{s_3} \cdot e(s_{cb}, A \cdot g_2{}^{c_r}) \in \mathcal{G}$. Finally, calculate the hash using

R'_c as an additional concatenated hash input value $c'_s = Hash(R'_A|R'_X|Ver|R'_c) \in \mathbb{Z}_p$. By checking that $c_s = c'_s$, the verifying peer can now trust the PPK and the credential information $CRED$.

The proof was derived the following way:

$$\left(s_{cb}{}^{1/r}\right)^{b+c_r} = h_1 h^x$$

$$s_{cb}{}^{b+c_r} = (h_1 h^x)^r$$

$$e(s_{cb}, g_2{}^{b+c_r}) = e(h_1 h^x, g_2)^r$$

$$e(s_{cb}, B \cdot g_2{}^{c_r}) = e(h_1, g_2)^r \cdot e(h, g_2)^{rx}$$

The credential signature is not directly traceable, however, since it is bound to the peer's private key, which is proven using the PPK_{sig}, the owner of a credential can always be found out by the CA. The proposed solution of the integration of a credential signature proof into the PPK_{sig} proof is the most efficient when the credentials are always the same. When another credential is attached to a message, the verifying peer has to revalidate the whole PPK_{sig}, since the proof has changed. In cases where different credentials are required, it can be more efficient to separate the proofs. The proof of the PPK_{sig} would be the original one and the proof of the credential would consist of R_X and R_C, so that $c_c = (R_X|R_c) \in \mathbb{Z}_p$. R_X is required to assure that the certificate is bound to the peer's private key x. The proof of this certificate would consist of $(c_c, s_1, s_3) \in \mathbb{Z}_p$. Both solutions also support the concatenation of multiple credentials. For each credential signature a separate R_C has to be computed and concatenated to the hash input. For instance, for n credentials $c_s = (R_A|R_X|Ver|R_{c1}|R_{c2}| \dots |R_{cn}) \in \mathbb{Z}_p$.

3.3 RSU Authentication and Authorisation

The GSIS scheme [LSH07], contains with its RSU authentication and authorisation scheme an efficient non-pseudonymous PKI enhancement. Since RSUs have no privacy requirements, PPKs are not necessary, and a more efficient scheme can be used. Conveniently, the GSIS RSU authentication can be adapted to the PPKI. However, this implementation uses the scheme only to sign the RSU's public key. Like regular peers, RSUs use efficient ECC based signatures for messages. The GSIS scheme uses an identity-based signature, in which the hash of the RSU's ID together with a credential string serves CS as a public key. In this adaption, the credential string and the RSU's public key is also signed by the *CA*. The *CA* not only signs the RSU ID and an optional credential string, but also the RSU's public key.

RSU Initialisation and Registration

For identity-based signatures, it is necessary that the RSU certificate key is created at the CA: **1.** RSU creates its private key $x \in \mathbb{G}_1$ and its public key $y = h^x$ for message signatures. **2.** RSU generates a proof s_{id} for ownership of its identifier ID, and a valid binding of its root public key x, ID and the credential string CS. **3.** *CA* verifies s_{id} and creates the unique RSU identifier $z = Hash(ID|y|CS) \in \mathbb{Z}_p$ **4.** *CA* creates the RSU's

certificate key: $t_r = {g_1}^{1/(a+h_{RSU})}$ using the CA's private key a. **5.** RSU calculates accelerator $v = e(g_1, g_2) \in \mathcal{G}$. **6.** RSU stores x,y,z,t_r and v.

RSU Public Key Signature Generation and Verification
In contrast to normal peers, this signature only needs to be recalculated after a version update. **1.** RSU creates the signature using the following procedure: Firstly, create random element $x_r \in_R \mathbb{Z}_p$. Secondly, compute the hash $c_r = Hash(R_r|Ver) \in \mathbb{Z}_p$ using $R_r = v^{x_r} \in \mathcal{G}$. Finally, compute certificate element $s_r = {t_r}^{x_r+c_r} \in \mathbb{G}_1$. **2.** A verifier can validate the signature $PK_{sig} = (c_r, s_r, y, Ver)$ by computing $z' = Hash(ID|CS|y)$, $R_r' = e(s_r, {g_2}^{z'} \cdot A) \cdot v^{-c_r} \in \mathcal{G}$, $c_r' = Hash(R_r'|Ver)$ and by verifying that $c_r = {c_r}'$.

RSU Message Signature Generation and Verification
RSU message signatures are created using the same procedure as for regular peers using the RSU's private key x the public y and the generator h.

RSU Revocation
It is very important that RSU can be revoked using the same method as used for peer revocation. Otherwise two revocation lists have to be used and the advantage of the update method would be lost. For that reason, the unique identifier z of an RSU is compatible to that of a peer. The *CA* can use the same revocation procedures and publish a new revocation data $(Ver, \tilde{g}_1, \tilde{g}_2, \hat{z})$ to the revocation list.
The RSU uses the same version update protocol as a peer, but instead of $\widetilde{t_g}$ it calculates its new private key $\widetilde{t_r} = (\tilde{v}/t_r)^{1/(z-\hat{z})}$ with the new accelerator $\tilde{v} = e(\tilde{g}_1, \tilde{g}_2)$. It can be seen again, that a revoked RSU cannot perform the calculation of $\widetilde{t_r}$.

4 Analysis

The two central questions for the utilisability of the PPKI are: how much the signature sizes of the PPKI differ from a classical PKI scheme and whether the PPKI has a sufficient performance. Both depend on the selected key lenghts. As explained above, the groups $\mathbb{G}_1$ and $\mathbb{G}_2$ are subgroups of an elliptic curve over finite fields, the third group $\mathcal{G}$, however, is a subgroup of a finite field. Hence, two cryptographic key lengths are relevant: Firstly, for the size of p for elliptic curve based cryptography and secondly, the size of the field p^k for the multiplicative subgroup $\mathcal{G}$. To provide security until the year 2030, key lengths for p of 224 bit and for p^k of 2432 bit are recommended, according to ECRYPT II [Sm11]. This requires the implementation of Barreto-Naehrig elliptic curves [BN06], with the parameter $k = 12$, which results in a length of 2688 bit for p^k.

4.1 Signature and Credential Size

A peer's message signature package consists of the message signature itself, the PPK, the PPK signature and the version identifier. Table 1 shows how the signature size is determined. Due to *point compression* for elliptic curves, elements of $\mathbb{G}_1$ can be reduced to the x-value plus an indicator bit for the y-value. An ECC-based solution would consist of two

elements for the message signature, one element for the peer's public key and two elements for the public key signature. This size would be $5 \cdot log_2(p)$, which is 1120 bit using a key size of 224 bit. The PPKI's signature is 66 % larger than an ECC-based one. The non-pseudonymous RSU signature, as shown in Table 2, has about the same size as a regular ECC-based one. The difference depends on the size of the version identifier and the credential string. The *RSU-ID* is already part of the Car-2-Car protocol. Table 3 shows the determination of the authorisation certificate. It is easy to see that the *AA* public key $B \in \mathbb{G}_1$ is very large compared to the other elements, although due to *six-to-one point compression* of Baretto-Naehrig curves [BN06] it is only double the size instead of twelve times. The other sizes are comparable to an ECC-based solution. Since the public keys of authorities do not change very often, a possible solution to avoid this problem is to distribute public keys separately. The credential then would only need to refer to the required key using a small identifier.

Table 1: Components and sizes of the PPKI peer signature

Signature Parts	Elements	Relative Size	Example for $p \triangleq 224$ and $ver \triangleq 32$ (bit)
Message Signature	$c_{sm} \in \mathbb{Z}_p$ $s_{sm} \in \mathbb{Z}_p$	$\log_2(p)$ $\log_2(p)$	224 224
Pseudonymous Public Key	$t \in \mathbb{G}_1$ $t_y \in \mathbb{G}_1$	$\log_2(p) + 1$ $\log_2(p) + 1$	225 225
Pseudonymous Public Key Signature	$c_s \in \mathbb{Z}_p$ $s_1 \in \mathbb{Z}_p$ $s_2 \in \mathbb{Z}_p$ $s_3 \in \mathbb{Z}_p$	$\log_2(p)$ $\log_2(p)$ $\log_2(p)$ $\log_2(p)$	224 224 224 224
Version	ver	$\log_2(ver)$	32
		$8\log_2(p) + \log_2(ver) + 2$	1826

Table 2: Components and sizes of the PPKI RSU signature

Signature Parts	Elements	Relative Size	Example for $p \triangleq 224$, $ver \triangleq 32$ and CS $\triangleq 64$ (bit)
Message Signature	$c_{sm} \in \mathbb{Z}_p$ $s_{sm} \in \mathbb{Z}_p$	$\log_2(p)$ $\log_2(p)$	224 224
Public Key	$y \in \mathbb{G}_1$	$\log_2(p) + 1$	225
Public Key Signature	$c_r \in \mathbb{Z}_p$ $s_r \in \mathbb{Z}_p$	$\log_2(p)$ $\log_2(p)$	224 224
Version Cred. String	ver CS	$\log_2(ver)$ $log_2(CS)$	32 64
		$8\log_2(p) + \log_2(ver) + 2$	1217

Table 3: Components and sizes of the PPKI credential

Signature Parts	Elements	Relative Size	Example for $p \triangleq 224$ and CS $\triangleq 64$ (bit)
Cred. String and Signature	CS	$\log_2(CS)$	64
	$s_{cb} \in \mathbb{G}_1$	$\log_2(p) + 1$	225
AA Public Key with *Root CA*'s Signature	$B \in \mathbb{G}_2$	$\log_2(p^2) + 1$	449
	$c_{rb} \in \mathbb{Z}_p$	$\log_2(p)$	224
	$s_{rb} \in \mathbb{Z}_p$	$\log_2(p)$	224
		$\log_2(CS) + 5\log_2(p) + 2$	1186

4.2 Performance

The main performance constraints in this protocol are the operations in $\mathcal{G}$ and the pairing e. Since signing and validating are the operations most often performed on the peer side, they will be subject to this benchmark. Compared to the validation process, the signature process misses one pairing operation. This means that the validation process is computationally more expensive than the signature creation. Hence, the sign operation can be neglected, especially since it is only rarely used. Message signature operations are performed in $\mathbb{G}_1$ and are equal to a common ECC-based Schnorr implementation. The time required for a message signature validation serves as a reference for the PPK signature performance. The same applies to the authorisation credential verification. Since no other suitable pseudonymous authorisation scheme was found, the only comparable solution is short-term ECC-based certificates, similar to the short-term public keys. Again, the message verification time is taken as a reference. The benchmark was performed on a 2.7 GHz Intel i7 CPU As Table 4 shows, the validation times and ratios using a 254-bit on Berito-Nehrig curve using a performant Optimal Ate pairing library [BG10]. Since automotive implementations will not use PC CPUs but cost and energy efficient processors or application-specific integrated circuits (ASICs), the ratios of the different verification processes are more important than the measured times.

Table 4: Performance Comparison of PPKI Operations on Berito-Nehrig curve with $p \triangleq 224$ bit

Operation	Time / ms	ratio
Message Verification	0.426	1
PPK Verification	3.956	9.27
RSU PK Verification	2.052	4.82
Cred. Verification	3.004	7.05

The result shows that the PPK signature verification is about *ten times* slower than the message verification process. The slowdown usually affects only the first message of a new peer, because only at this point a PPK verification is required. Subsequently, until the next PPK change, only message signatures need to be validated. Nevertheless, this is a situation requiring further investigation: since a PPK change is only reasonable when all peers within a certain range perform it at the same time, the performance can become critical. It may help to specify the maximum size of the group of peers which execute the PPK change. The RSU public key verification is *twice as fast* as the PPK verification.

A pseudonymous authorisation credential verification is *seven times* slower than short term ECC based certificates, however still faster than a PPK verification. Safety relevant messages that contain an authorisation credential require both to be verified: the PPK and the authorisation credential. Together it would take 6.96 ms for the verification, which is *eight times* more than two ECC-based verifications.

5 Conclusion

In this work, it could be shown that the PPKI does indeed meet the challenges of Car-2-Car communication. Message authenticity, integrity, and non-repudiation are fulfilled equally. It is possible to set up a hierarchical PKI architecture with regional CAs and multiple *AAs*.

The new developed pseudonymous authorisation scheme allows to integrate trusted authorisation authorities into the PPKI. They enable peers to use special safety messages or commercial services like tolling, information or entertainment services. Similarly to the PPKI authorisation it allows peer side pseudonymous credential generation with about the same performance requirements. Without such a solution, the amount of pseudonymous authentication credentials for multiple services can become enormous, and the issuing process would be computational expensive, since each pseudonymous public key requires a unique authentication credential. On-demand issuing services [AGL13] may be practically applicable for some commercial services, however they would be much more complex then the PPKI solution. An important privacy feature of the new pseudonymous authorisation scheme is the optionally "blinded" registration process that prevents the authorisation authorities to link the issued credentials to a peer. If a customer pays a tolling authority for a one-year certificate, the authority does not need to know the peer's identity. The privacy of this process can be compared to the process of buying a tolling vignette.

In contrast to the classical PKI, the PPKI allows the peers to change their pseudonym at any required frequency. In addition, classical PKI solutions require an Internet connection to distribute pseudonyms and authorisation certificates and maybe also to provide CRLs. After the initial registration at a CA and AA, a PPKI peer can rely only on Car-2-Car communication. This makes the solution robust and user-friendly. Also, the revocation process is straightforward: peers are forced to distribute and import revocation data, otherwise they can no longer take part in the communication.

Nevertheless, those features come with additional computational costs for key and credential validation. A PPKI hardware solution will probably be more expensive, especially since a pairing implementation is more complicated than a plain ECC based one. A quantitative determination of costs and benefits cannot be made since costs could be reduced by mass production of ASICs. Although the system in the car becomes more sophisticated, it carries out tasks that would otherwise be located on the infrastructure side. From a qualitative point of view, the PPKI scales better, the communication overhead is reduced and administrative processes become easier to manage.

References

[AGL13] Alexiou, N.; Gisdakis, S.; Laganà, M.; Papadimitratos, P.: Towards a secure and privacy-preserving multi-service vehicular architecture: WOWMOM 2013, IEEE

[BN06] Barreto P.; Naehrig M.: Pairing-Friendly Elliptic Curves of Prime Order: Selected Areas in Cryptography - SAC 2005; Lecture Notes in Computer Science: Springer, 2006

[BF00] Boneh D.; Franklin M.: Identity-Based Encryption from the Weil Pairing: Advances in Cryptology, Springer Berlin Heidelberg, 2001

[BG10] Beuchat, J.; Gonzáles-días, J.; Mitsunari, S.; Okamoto, E.; Rodríguez-henríquez F.; Teruya, T.: High-speed Software Implementation of the Optimal Ate Pairing over Barreto-Naehrig Curves: Proceedings of the 4th International Conference on Pairing-based Cryptography, Pairing'10, Springer-Verlag Berlin, Heidelberg 2010

[CM09] Cao Z.; Markowitch O.: Security Difference between DSA and Schnorr's Signature: Networks Security, Wireless Communications and Trusted Computing: IEEE, 2009

[ES12] Escrypt: Efficient Public Key Infrastructure (PKI) Solutions for Embedded Systems: Website Report: https://www.escrypt.com/company/single-news/detail/efficient-public-key-infrastructure-pki-solutions-for-embedded-systems/

[ETSI1] ETSI Specification 102 940: Intelligent Transport Systems; Security; ITS communications security architecture and security management: European Telecommunications Standards Institute, 2012

[ETSI2] ETSI Specification 102 941: Intelligent Transport Systems; Security; Trust and Privacy Management: European Telecommunications Standards Institute, 2013

[ETSI3] ETSI Specification 102 636-4-1: Intelligent Transport Systems; Vehicular communications; GeoNetworking; Part 4: Geographical addressing and forwarding for point-to-point and point-to-multipoint communications; Sub-part 1 Media-Independent Functionality: European Telecommunications Standards Institute, 2011

[GPS06] Galbraith S.; Paterson K.; Smart N.: Pairings for Cryptographers: Cryptology ePrint Archive, Report 2006/165, 2006

[JPBC] Angelo De Caro: The Java Pairing-Based Cryptography Library: Università degli Studi di Salerno

[LLP08] Lee E.; Lee H.; Park C.: Efficient and generalized pairing computation on abelian varieties: Cryptology ePrint Archive, Report 2008/040, 2008.

[LSH07] Lin, X.; Sun, X.; Ho, P.; Shen, X.; GSIS: A Secure and Privacy-Preserving Protocol for Vehicular Communications: IEEE Transactions on Vehicular Technology, 56, 2007

[Ly07] Lynn, B: On the implementation of pairing-based cryptosystems: Stanford University, 2007

[Sm11] Smart, N.: ECRYPT II Yearly Report on Algorithms and Keysizes: European Network of Excellence in Cryptology II, 2011

[Pa08] Papadimitratos, P. et al: Secure vehicular communication systems: design and architecture: IEEE Communications Magazine, Vol 46, Issue 11: IEEE Press, Piscataway: 2008, pp 100-109

[Sc89] Schnorr C.: Efficient Identification and Signatures for Smart Cards, Advances in Cryptology, Springer Berlin Heidelberg, 1989

[Ve01] Verheul, E.: Self-Blindable Credential Certificates from the Weil Pairing: ASIACRYPT '01, Proceedings of the 7th International Conference on the Theory and Application of Cryptology and Information Security: Advances in Cryptology, Springer-Verlag London 2001; pp. 533-551

[Ve10] Vercauteren F.: Optimal Pairings: IEEE Transactions on Information Theory: 2010; pp 455-461

[Ze06] Zeng, K.: Pseudonymous PKI for Ubiquitous Computing: EuroPKI 2006, LNCS 4043. Springer-Verlag Berlin Heidelberg 2006; pp. 207-222

Towards an Information Security Framework for the Automotive Domain

Benjamin Glas, Jens Gramm, Priyamvadha Vembar

Bosch Center of Competence Security
ETAS GmbH
Borsigstraße 14
70469 Stuttgart
{ benjamin.glas | jens.gramm | priyamvadha.vembar }@etas.com

Abstract: Driven by increasing connectivity, complexity, and distribution of functions in vehicles, automotive security is steadily gaining attention. While in other domains there is a harmonized notion on security requirements and there are even broad information security management standards, in the automotive domain there is so far no common structured approach to achieve system security. At the same time, the automotive domain shows some special properties that differentiate it from other domains and prevent direct application of available approaches.

In this work, we take a step towards an automotive information security framework. We provide an overview of existing system security and security development standards from related domains and provide motivation for harmonization on one hand and a sector-specific standard on the other. We outline core ideas and elements of a possible framework for automotive security engineering, illustrate them with examples, and put them in context with existing related information security standards.

1 Introduction

In modern cars, we observe a trend towards growing interconnectivity of embedded systems, along with integration of mobile devices and new interfaces to external systems and the Internet. The paradigm of private networks in the car is no longer valid. Formerly isolated islands become part of an interconnected and open automotive system, bringing advantages for functionality, comfort and cost.

However, growing interconnectivity comes with new implications on the dependability of automotive electric/electronic (E/E) systems. This has already been addressed for functional safety – i.e. absence of unreasonable risks (for health and life) with a focus on risks caused by random hardware failures and systematic failures; functional safety is an established discipline in automotive engineering. It is a comparably new insight that also (information) security – i.e. the quality of being robust against intentional manipulation by a Human adversary - is a necessary precondition for a dependable system and has to be addressed in the engineering approach for automotive systems.

On the one hand, we see a growing attack surface due to an increasing number of interfaces and a growing interconnectivity of functions on automotive ECUs. On the other hand, we see practically demonstrated attacks on automotive systems [CMK11,KCR10,HKD08]. For these reasons, automotive security, i.e. information security in automotive E/E systems, is gaining attention in recent years.

Looking for available guidelines, we find a number of information security standards, e.g. ISO 27001 [ISO13], the German IT-Grundschutz [BSI08], and Common Criteria [CC12]. But how do these standards help us to anchor security in automotive product development processes? In fact, we claim that application of general standards requires tailoring them to the specific needs of the automotive domain. Moreover, we see a need for an industry-defined sector-specific automotive security standard. There are other domains, for which adapted security frameworks are proposed or on the way, e.g. IEC 62443 for industrial automation [IEC] or adaptations for railway systems [BBM12,Br14]. For the automotive domain, no tailored and accepted security framework exists.

In this work, we address the question for an automotive-specific information security standard with the following goals:

- extend current automotive quality systems to include security and anchor security in automotive product development processes to ensure security of automotive systems,
- provide opportunity to identify and manage risks to information and systems assets in automotive products,
- achieve confidence and assurance between business partners, and allow an independent review and assurance on information security practices in automotive products.

Our scope is information security in automotive E/E systems. Cross-relations, in this context, between functional safety and security are a complex topic currently under investigation, e.g. see [GGH15], and are beyond the scope of this contribution.

In this paper, we propose a standard framework for automotive security, combining elements of existing IT security standard frameworks and demonstrating how to apply them in the automotive context. We continue and extend proposals made in [GW14,GGP14].

2 Normative Background

In this section, we provide an overview on existing information security and automotive standards that are referenced in the remainder of this work.

- **ISO/IEC 27000 family**, growing family of information security standards. At its core, the ISO 27001 standard [ISO13] specifies an information security management system to bring information security under management control. The standard focuses on classical IT systems and on information management in organizations.

- **IT Grundschutz** is issued by the German Federal Office for Information Security Technology ("Bundesamt für Sicherheit in der Informationstechnik", BSI) in four main parts (BSI Standards 100-1 to -4) with auxiliary online documentation ("IT-Grundschutz-Kataloge"), see [BSI08] as starting point. It has the goal to standardize the identification and implementation of security measures for corporate IT systems, i.e. focus is information management in organizations.

- **ISO 15408/Common Criteria**, coming in three parts, see [CC12] as starting point. The Common Criteria for Information Technology Security Evaluation form an international standard framework for computer security certification. The focus of Common Criteria is on assurance, i.e. assurance to be compliant with a claimed security target. As an element to enable harmonization of security across product classes, Common Criteria features Protection Profiles.

- **ISA99/IEC 62443 family**. IEC 62443 is a family of standards (in development, with 11 planned parts), see [IEC] as starting point, providing procedures for implementing secure Industrial Automation and Control Systems (IACS). For a closer look on this standard family see Section 3.

In summary, we see that there already exists a number of IT security standards. However, the generic ones focus on (corporate) IT systems (ISO 27000, IT Grundschutz) or assurance (Common Criteria). There are first examples of sector-specific information security standards in development, e.g. IEC 62443 and adaptations to railway systems [BBM12,Br14]. A comparable automotive-specific framework does, so far, not exist.

In the area of functional safety, we have seen how a domain-specific standard was introduced to address particular requirements of the automotive sector.

- **ISO 26262** [ISO11] defines functional safety for automotive equipment applicable throughout the lifecycle of all automotive electronic and electrical safety-related systems. The ISO 26262 standard for automotive systems was inspired by the more generic functional safety standard IEC 61508.

We propose to learn from existing standards to shape an industry-defined sector-specific automotive security standard.

3 Overview on IEC 62443 Standard

We identify the IEC 62443 series to be of particular interest, as it provides a good example for the development of a domain-specific security framework. This section provides an overview on elements of this standard family that will be referred to later-on. We particularly focus on the use of security levels in IEC 62443 [IEC13].

According to IEC 62443, the scope of detailed technical and program requirements shall be derived from a small set of "foundational requirements". More concretely, IEC 62443 specifies seven foundational requirements:

- identification and authentication control,
- use control,
- system integrity,
- data confidentiality
- restricted data flow,
- timely response to events, and
- resource availability.

For a given product, the security objectives of this product are characterized by attributing a security level for each foundational requirement. In this way, a product is characterized by a security level vector, with one security level (SL) assigned to each foundational requirement. SLs are assigned according to attacker capability and motivation, which a product is expected to be robust against, as follows:

- SL1: casual or unintended,
- SL2: simple means, i.e. low resources, generic skills and low motivation,
- SL3: moderate means, i.e. moderate resources, moderate motivation,
- SL4: sophisticated means, i.e. extended resources and high motivation.

IEC 62443 introduces a useful distinction between different viewpoints on security levels. A system integrator derives, based on a risk analysis for his overall system, a *target* security level for a particular component. The supplier develops and produces components offering a *capability* security level. Integrated into the overall system, depending on its configuration, the components realize an *achieved* security level.

IEC 62443-3-3 provides a catalogue of security controls, organized by foundational requirements and security levels. This catalogue provides a domain-specific, minimum set of requirements to reach progressively more stringent security levels. It includes system requirements, describing the security measures required to reach a security level for a foundational requirement.

4 Automotive Systems

In Figure 1, we present an illustration of future architecture of the E/E system in a modern car.

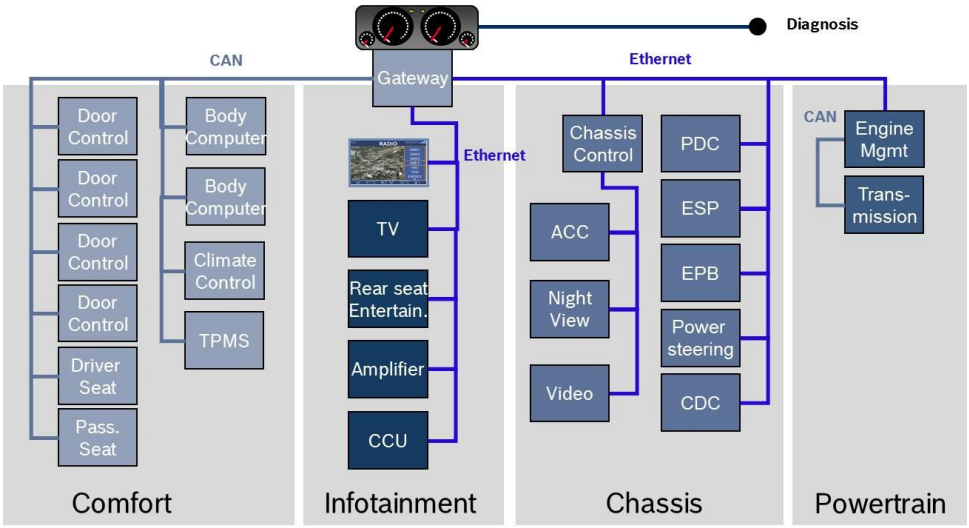


Figure 1: Exemplified architecture of an automotive E/E network.

In the following, we identify several characteristics of automotive systems in order to point out differences to classical IT systems.

- **Highly distributed**: Automotive systems are highly distributed systems with up to 100 single electronic control units (ECUs), connected via a number of different established automotive-specific bus systems.
- **Constrained devices**: Many components in the system have resource limitations, e.g. regarding storage and performance.
- **Real-time and architecture**: Timing is critical for many automotive systems, ECUs are often tailored to specific hardware.
- **Safety relevance**: Often components are part of safety relevant control systems and have real time requirements and high impact on passenger safety.
- **Untrusted environment**: The systems are in the field and not in a controlled and trusted environment, adversaries do potentially have physical access.
- **Constrained patchability**: Deployed systems are in the field without possibility to enforce software patches, and we observe long release times for safety-relevant software due to extensive testing requirements.
- **Longevity**: Automotive systems are expected to work reliably and with only sporadic surveillance and maintenance for up to 20 years in the field.

For embedded systems as they are used in the automotive domain, we see, on the one hand, specific threats not present for classical IT servers, e.g. adversaries with physical access to devices. On the other hand, we observe specific resource constraints, which have to be considered in the choice of security controls.

In the following, our target object here are "products" which may be single ECUs (e.g. engine control units) in the system but also functions involving several components in the system (e.g. adaptive cruise control).

5 Information Security Approach for the Automotive Domain

Security processes can be applied in a company-specific way, anchoring security in the company's product development processes. Execution of these processes is a product-specific activity. For example, security requirements are derived for the specific product, it is decided for this specific product how they are implemented and tested. But how do we ensure that all products of one product class come with a comparable level of security? How do we ensure an industry-wide harmonization on expected security functions? How do we ensure a common understanding between system integrators and suppliers?

In this section, we present elements forming a framework for automotive security (illustrated in Figure 2). We propose security profiles as key elements to provide harmonization of an industry-wide security framework.

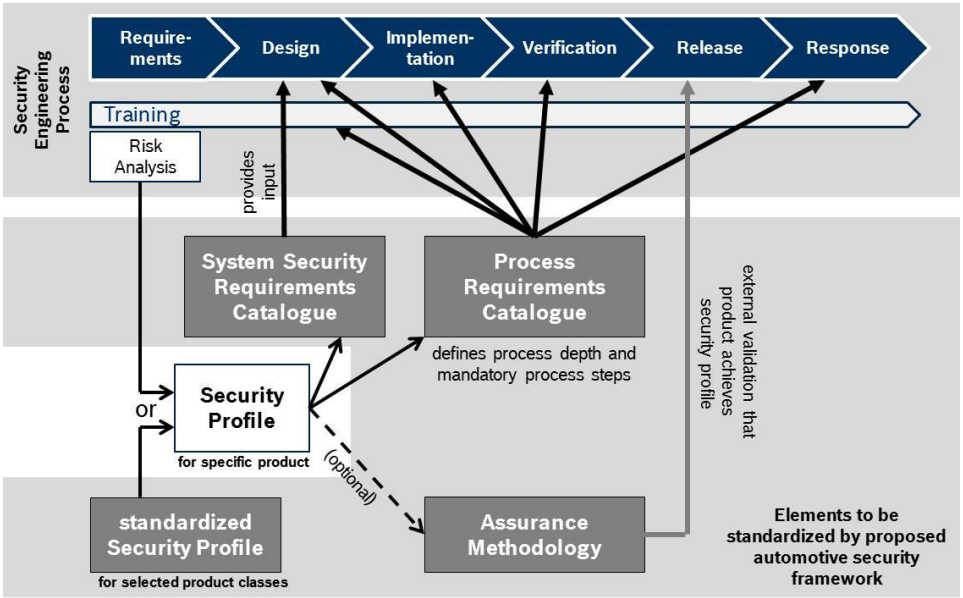


Figure 2: Illustration of the proposed security framework.

In the following, we, first, outline the scope of a security engineering process as the basis for a security framework. We, then, present

- Security Profiles

as an anchorpoint to allow harmonization – within a company, between business partners, as well as industry-wide - of security requirements, along with dependent framework elements, namely

- System Requirements Catalogue(s) for the automotive domain,
- Process Requirements Catalogue,
- Assurance Methodology.

Notably, we only identify these elements as ones suitable for an automotive security framework, outlining their scope and providing examples, but do, in this work, not provide the content of these documents.

Security Engineering Process

A Security Engineering Process defines (mandatory and recommended) activities for information security risk management in each phase of the product development cycle. To provide a concrete example for different phases to consider, we refer here to phases of the Microsoft Security Development Lifecycle: Training, Requirements, Design, Implementation, Verification, Release, Response.

As example activities of a Security Engineering Process, we highlight the derivation of security requirements for a product. This is done by, first, identifing and analyzing the information security risks of a product, and, second, to define appropriate security controls, in the Requirements and Design Phases of product development:

- **Risk Analysis**. For identifying information security risks, we consider who (attackers) could introduce a fault under which circumstances (vulnerabilities), why, how and when he would do it (attacker goals and attack) and what and whom this would affect (assets and security stakeholders). If there is an attacker, who might exploit a given vulnerability to carry out a certain attack against a certain asset that is of value for a stakeholder, this combination of attacker, vulnerability, attack and asset is called a threat.

  From identified threats, we arrive at risks by assessing the threat. Herein, a risk is typically determined by two factors, namely *Impact* caused by realization of a misuse case, and *Likelihood* by which a misuse case is realized. A concrete rating scheme is omitted here. Risks may be accepted, mitigated or prevented, or delegated. Result of a Risk Analysis is a set of information security risks which are to be addressed by the product.

- **Security Concept**. A Security Concept specifies, on a design level, security controls by which risks are to be mitigated or prevented in the product. A

Security Concept may be developed and refined over different levels of abstraction.

Comparable elements of security programs, e.g. risk analysis methodologies, are provided by ISO 27005, IT Grundschutz, and IEC 62443.

In the following, we present extensions to a Security Engineering Process, with the goal of harmonization and standardization.

Security Profiles

A Security Profile states, in a compact way, the security goals for a product class or a given collection of systems. It specifies high-level security requirements for a product. We propose to specify Security Profiles by combining ideas from several of the presented standards.

First, we follow the idea of using Security Levels (as used in IEC 62443 and IT Grundschutz), to classify the risk (or "protection need" in IT Grundschutz language) of a product, component or function. Security Levels are taken from a defined set, we assume four levels for illustrational purposes. (It will be outlined in the following how to obtain a Security Level for a product.)

Second, we follow the idea of using *Security Level Vectors* (as used in IEC 62443) by determining a Security Level for each of a set of "foundational requirements". "Foundational requirements" are used here in the same way as they were introduced in Section 3. Foundational requirements are a small set of basic security requirements or "security goals", respectively. IEC 62443-1-1 [IEC] introduces a set of seven foundational requirements (see Section 3). We propose to adapt the set of foundational requirements for the automotive domain. As an example, we propose, for automotive systems, an illustrative set of foundational requirements presented in Table 1. A small example of a Security Level Vector is shown in Figure 3. An overall Security Level for the product may, then, be derived by taking the maximum over the entries of a Security Level Vector.

Third, we use the idea of determining a Security Level Vector *for each misuse case* (in analogy to a Hazard&Risk Analysis according to ISO 26262), and determining the entries of the Security Level Vector by rating *risk levels* for each foundational requirement.

To clarify terminology here, "misuse cases" describe negative use cases. The negative scenario is not desired by the stakeholders but by a hostile adversary - using a system to one's advantage. For automotive ECUs, a typical misuse cases may be "an external adversary endangers car safety".

For each misuse case, a risk level is determined for each foundational requirement. Each entry expresses the risk imposed by the misuse case on the foundational requirement. A Security Level Vector for the product is, then, obtained by maximizing over all misuse

cases, for an illustration see Figure 4. Risks levels are derived based on two factors, namely impact and likelihood (see Risk Analysis section). As this rating is already part of a Risk Analysis, a Security Profile can be directly derived from a Risk Analysis done for a particular product or system.

A Security Profile is complemented by a specification of its target and scope. In summary, a Security Profile provides a compact specification of the security goals of a product, with different levels of abstraction: A Security Level for a quick overall assessment, a Security Level Vector for an assessment refined to foundational requirements, and a more detailed risk analysis on the level of misuse cases. We argue that each of these abstraction levels is of use in certain circumstances. For example, decisions on process activities (e.g. testing depth) may be based on the Security Level alone. Decisions on system security requirements, however, may apply to certain assets only.

Integrity	Firmware Integrity	*Firmware integrity* is the property of safeguarding the accuracy and completeness of the firmware stored on devices. Integrity refers to trustworthiness of data or resources, and is normally phrased in terms of preventing improper or unauthorized change. It includes *data integrity* and *origin integrity*. Data integrity refers to the correctness and trustworthiness of data or content of information. Origin integrity refers to the integrity of source of data (often called authenticity), i.e. that data are original as generated by an authorized source or author.
	Communication Integrity	*Communication integrity* is the property of safeguarding the accuracy and completeness of data in transit. It includes *data integrity* and *origin integrity* but also *freshness*, i.e. a protection against replay of recorded data.
		…
Confidentiality	Data Confidentiality	*Data confidentiality* is the property that information is not made available or disclosed to unauthorized individuals, entities or processes. This may also include privacy aspects. Here, data confidentiality is understood to apply to data processed by the product.
	IP Protection	*IP Protection* (IP=intellectual property) refers to data being protected against unauthorized copying. Here, IP protection is understood to apply to software/firmware of the product.
		…
Availability		*Availability* refers to being accessible and usable upon demand by an authorized entity.

Table 1: Example for foundational requirements suited to automotive systems.

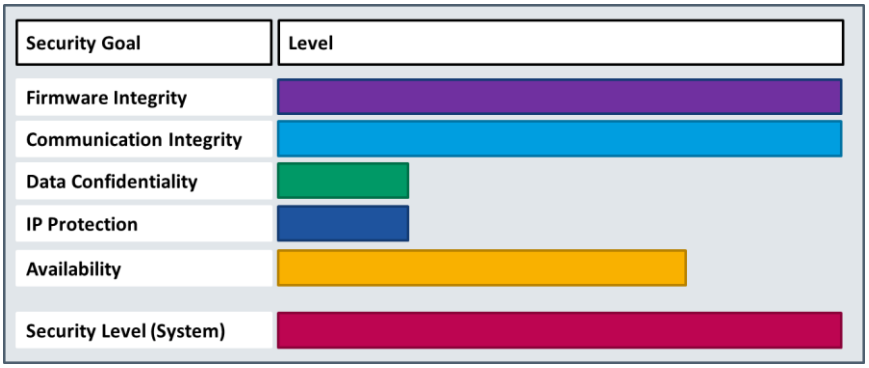


Figure 3: Example for a Security Level Vector of a safety-relevant automotive ECU.

	System Integrity	Communication Integrity	...	Availability
Misuse Case 1	$Risk_{SI,1}$	$Risk_{CI,1}$	...	$Risk_{A,1}$
Misuse Case 2	$Risk_{SI,2}$	$Risk_{CI,2}$	...	$Risk_{A,2}$
...				
Misuse Case *n*	$Risk_{SI,n}$	$Risk_{CI,n}$	...	$Risk_{A,n}$
Security Level Vector	$\max_{j=1...n} Risk_{SI,j}$	$\max_{j=1...n} Risk_{CI,j}$	...	$\max_{j=1...n} Risk_{A,j}$

Figure 4: Illustration on the derivation of Security Level Vectors from a Risk Analysis.

The advantages of Security Profiles are as follows:

- They provide a compact representation of the security goals (a "security problem definition" in the Common Criteria language) of a product. In this way, they help to establish a common understanding – within a company, between business partners, or even within an industry.

- They capture the risk associated with a component. Their definition is a step towards an economic way of dealing with security. They help to, then, scale security controls and security-related process activities in accordance to the identified risks, i.e. to spend most effort where the highest risks are.

- Security Profiles may be agreed on and standardized for certain product classes. A standardized security profile saves a detailed risk analysis for every new product instance. It provides an accepted standard on security properties of a product.

Security Profiles as introduced here relate to elements of existing standards in the following ways:

- The Security Profile follows the idea of Security Level Vectors as they are used in the IEC 62443 standard.

- IT Grundschutz [BSI08] includes a concept related to Security Levels as follows. An application is assessed by rating, for each use application use case, a security level (here called "protection need level") for the security goals confidentiality, integrity and availability.

- Formulating the Security Profiles on the level of misuse cases follows the approach used by the ISO 26262 standard. There, a Harzard&Risk Analysis is done on the level of hazards, to rate risks and to analyze safety goals associated with a product. Safety goals are captured by one of four "Automotive Safety Integrity Levels".

- A standardized Security Profile serves a similar purpose as a Protection Profile according to Common Criteria. However, the focus of Common Criteria lies on assurance. It does not specifies a requested Evaluation Assurance Level (EAL), indicating the depth and rigor of an security evaluation.

System Security Requirements Catalogue

The goal of the Security Requirements Catalogue is to define a domain-specific, minimum set of requirements to reach increasingly more stringent security levels. They are detailed system requirements, describing required security measures, associated with the foundational requirements. For an example see Table 2.

The System Security Requirements Catalogue specifies abstract functional requirements. It does not make a security concept obsolete. It formulates minimum requirements to be then detailed out by a Security Concept, potentially complemented by additional requirements. The catalogue's requirements form a frame for the Security Concept which, then, describes how these requirements are to be met in the specific product. The System Security Requirements Catalogue ensures that proven and standardized measures are taken based on a specific Security Profile. In this way, it ensures a minimum standard for a security concept and provides a standard set of security requirements to be implemented according to the state-of-the-art. Thus, it ensures a common understanding what is to be done when a specific Security Profile is agreed between two partners.

Notably, requirements in the catalogue have to be effective, practicable as well as economically feasible. For this reason, the catalogue is domain-specific and catalogues

for other domains are not directly transferable. Naturally, the System Security Requirements Catalogue has to be updated according to the state-of-the-art. In this way, it ensures a continuous improvement process and a progression of the state-of-art beyond system and company barriers.

It remains an open question to pose a System Security Requirements Catalogue for the automotive domain, for a case study see [GGP14].

Level	**Measure**	**Requirement**
0	None	None
1	Basic Protection	The system shall authenticate firmware by mechanisms which protect against casual or coincidental modification. The system shall grant access to internal data over all openly (from outside the device body) accessible interfaces only after successful authentication. …
2	Secure Reprogramming	The system shall provide the capability to accept only firmware updates with digital signatures.
	Remote Access Authentication	The system shall allow access over *remote* interfaces only after successful authentication with cryptographic mechanisms.
	…	
3	Local Access Authentication	The system shall grant write access *over local or vehicle-internal* interfaces only after successful authentication with cryptographic mechanisms.
	…	
4	Secure Boot	The system shall provide the capability to support verification of code, including the code implementing the software reprogramming function, on startup of the system. The verification shall be anchored in hardware.
	Physical Access Authentication	The system shall grant write access over *all* interfaces only after successful authentication with cryptographic mechanisms. This does include also interfaces accessible after opening the device body.
	…	

Table 2: Illustrative example for system security requirements, here for System Integrity, details see [GGP14]. System security requirements are accumulative for increasing levels, i.e. system requirements for a level also include requirements for preceding levels.

System Security Requirements as presented relate to elements of existing standards in the following ways:

- The System Security Requirements Catalogue directly follows the idea of the one proposed in IEC 62443-3-3. Requirements posed there are targeted towards industrial automation and control systems, however. A targeted catalogue for automotive systems is needed.

- System Security Requirements can be compared to the information security controls presented by the ISO 27002 standard and the "Maßnahmenkatalog" in IT Grundschutz. The security controls presented there, however, are focused to IT systems.

- System Security Requirements can be compared to the list of Security Functional Requirements (SFRs) which are defined in the Common Criteria standard. SFRs, however, are generic and not domain-specific. As one particular useful feature, the Common Criteria list of SFRs identifies dependencies between the requirements, i.e. where the correct operation of one function is dependent on another, e.g. the ability to limit access according to roles requires the ability to identify individual roles.

Process Requirements Catalogue

A Process Requirements Catalogue derives, from a security level, the process variant which is used to develop a product. The Process Requirements Catalogue specifies, in dependence on the security level of a product, the selection of mandatory security-related process activities. For example, this includes extent of documentation and control, testing scope and depth, and the extent of security monitoring required when products are in the field. An illustrative example is provided in Table 3.

As advantage, a Process Requirements Catalogue ensures a common understanding on process elements expected for products of a certain security criticality. Selection of process activities is not done arbitrarily. Instead, depending on the risk of a component captured by a Security Profile, a project manager derives required process measures from the Process Requirements Catalogue, following a transparent set of rules.

We pose it as an open question to standardize a Process Requirements Catalogue for the automotive domain.

Similar as a Process Requirements Catalogue, the ISO 26262 standard specifies process requirements depending on the attributed ASIL classification.

Assurance Methodology

Evaluation and certification serves to validate claims made about the target, providing assurance to a customer that claimed security goals are met.

	Level			
Phase	1	2	3	4
Training	no specific process requirements	- Security awareness for architects and developers	- Secure coding for developers.	- Trained security specialist in project team
Requirements and Design		- Threat and Risk Analysis - Security Concept	- Design principles/design requirements	
Implementation		- Compiler warnings	- Secure coding guidelines - Static code analysis	- Code reviews - Dynamic program analysis
Verification		- Acceptance tests for security requirements	- End-of-development penetration testing - Fuzz testing	- In-development defensive security testing
Release				- Final security review - Incident response plan
Monitoring			- General surveillance of vulnerabilities in underlying technologies	- Product-specific monitoring of vulnerabilities. - Online patch management process

Table 3: Illustrative example for Process Requirements Catalogue, with process requirements for different lifecycle phases, depending on security levels. Process requirements are accumulative for increasing levels, i.e. process requirements for a level also include requirements for preceding levels.

Common Criteria provides a detailed and established framework for security evaluation and certification. Due to efforts involved to reach a certification for meaningful levels of confidence, until today, the Common Criteria approach is mainly employed for high-security applications, e.g. smart cards. Downstripping the Common Criteria idea for the automotive context is also the idea in [KAS13], presenting Trust Assurance Levels (TAL) and a certification framework to attest and certify the trustworthiness of a communication partner in vehicle-to-vehicle or vehicle-to-infrastructure communication.

We propose to develop a dedicated Assurance Methodology for the automotive context, following the ideas of Common Criteria and IEC 62443, but considering particularities of automotive development processes and supply chains, and respecting practicality in the automotive context. An Assurance Methodology is, then, used as follows:

- Based on a component´s risk surface in a system, a system integrator requires for system components conformance to a target security profile. This may be a target security profile developed based on a risk analysis or a standardized security profile in analogy to a Common Criteria Protection Profile.

- A component supplier builds a product, claiming a capability security profile, adhering to defined process requirements (e.g. mandatory tests) and defined system security requirements.

- An external evaluator or certification body validates, along the Assurance Methodology, the achieved Security Profile in the system.

We note, however, that we consider assurance as an *optional* element of an information security framework. The goal of the framework proposed here is to improve automotive security. The focus of assurance is on attaining, by a certification process, confidence that claimed security goals are met, which is only a subaspect of the former.

6 Conclusion

In this work, we propose elements of a security framework for the automotive domain, in particular:

- Security Profiles for automotive product classes: Security Profiles provide a specification of the security goals for a product or a class of products. Standardizing a Security Profile establishes an industry-wide common understanding on the security goals for a product class.

- System Security Requirements Catalogue. A System Security Requirements Catalogue defines security requirements on the level of controls, in relation to security levels in a Security Profile. Given a specific Security Profile, the catalogue implies a (minimum) set of security requirements to be fulfilled by the product and, thus, helps to establish a common understanding on the security to expect from a product.

- Process Requirements Catalogue. A Process Requirements Catalogue defines, in relation to security levels in a Security Profile, mandatory activities to be taken in the development of a product. Given a specific Security Profile, the catalogue implies (minimum) requirements on the Security Engineering Process, and, thus, helps to establish a common understanding on the security-related “care” taken in the development of the product.

- Assurance Methodology (optional). An Assurance Methodology defines the measures to be taken by an external evaluator to assess the compliance of a product with a defined Security Profile.

In this work, we outline these framework elements, illustrate them with examples, and put them in context of existing information security standards. It remains an item of future work to elaborate case studies and detailed proposals for the identified framework elements.

References

[BBM12] H.-H. Bock, J. Braband, B. Milius, H. Schäbe. Towards an IT Security Protection Profile for Safety-Related Communication in Railway Automation. In Proc. SAFECOMP'12, LNCS vol. 7612, pages 137-148, Springer-Verlag, 2012.

[Br14] J. Braband. Towards an IT Security Framework for Railway Automation. In Proc. ERTS2 Congress, 2014.

[BSI08] BSI-Standard 100-2: IT-Grundschutz Methodology, Version 2.0. Bundesamt für Sicherheit in der Informationstechnik, 2008. Available via http://www.bsi.bund.de/grundschutz.

[CMK11] S. Checkoway, D. McCoy, B. Kantor, D. Anderson, H. Shacham, S. Savage, K. Koscher, A. Czeskis, F. Roesner, T. Kohno. Comprehensive Experimental Analyses of Automotive Attack Surfaces. In Proc. USENIX Security, 2011.

[CC12] Common Criteria for Information Technology Security Evaluation, Part 1: Introduction and general model. Version 3.1, Revision 4, 2012. Available via http://www.commoncriteriaportal.org/cc/

[GGH15] B. Glas, C. Gebauer, J. Hänger, A. Heyl, J. Klarmann, S. Kriso, P. Vembar, P. Wörz. Automotive Safety and Security Integration Challenges. In Proc. Automotive Safety&Security 2015, LNI No. 240, Springer, 2015.

[GGP14] B. Glas, J. Gramm, P. Vembar. Security Levels and System Security Requirements for System Integrity of Automotive ECUs. In Proc. ESCAR 2014.

[GW14] B. Glas, M. Wolf. Punktlandung für Security - Ein holistischer Ansatz für mehr Embedded Security in automobilen Systemen. In Automobil-Elektronik 02/14, 2014.

[HKD08] T. Hoppe, S. Kiltz, J. Dittmann. Security Threats to Automotive CAN Networks – Practical Examples and Selected Short-Term Countermeasures. In Proc. SAFECOMP'12, LNCS vol. 5219, pages 235-248, Springer-Verlag, 2008.

[IEC] IEC62443-1-1, Industrial communication networks – Network and system security – Part 1-1: Terminology, concepts and models, to appear.

[IEC13] IEC62443-3-3, Industrial communication networks – Network and system security – Part 3-3: System security requirements and security levels, 2013.

[ISO11] ISO 26262-2:2011: Road vehicles -- Functional safety -- Part 2: Management of functional safety, 2011.

[ISO13] ISO/IEC27001:2013, Information technology – Security techniques – Information security management systems – Requirements, International Organization for Standardization, 2013.

[KAS13] A. Kiening, D. Angermeier, H. Seudié, T. Stodart, M. Wolf. Trust Assurance Levels of Cybercars in V2X Communication. In Proc. 1st ACM Workshop on Security, Privacy & Dependability for Cyber Vehicles (CyCar), ACM Press, 2013.

[KCR10] K. Koscher, A. Czeskis, F. Roesner, S. Patel, T. Kohno, S. Checkoway, D. McCoy, B. Kantor, D. Anderson, H. Shacham, S. Savage. Experimental Security Analysis of a Modern Automobile. In Proc. IEEE Symposium on Security and Privacy, 2010.

Security Crash Test – Practical Security Evaluations of Automotive Onboard IT Components

Stephanie Bayer, Thomas Enderle, Dennis Kengo Oka[*)], Marko Wolf

ESCRYPT GmbH
Leopoldstraße 244
80807 München
Germany

[*)] ETAS K.K.
Queens Tower C-17F
2-3-5, Minatomirai, Nishi-ku
Yokohama, 220-6217 Japan

{stephanie.bayer; thomas.enderle; marko.wolf}@escrypt.com
dennis-kengo.oka@etas.com

Abstract: Modern vehicles consist of many interconnected, software-based IT components which are tested very carefully for correct functional behavior to avoid safety problems, e.g. that the brakes suddenly stop working. However, in contrast to safety testing systematic testing against potential security gaps is not yet a common procedure within the automotive domain. This however could eventually enable a malicious entity to be able to attack a safety-critical IT component or even the whole vehicle. Several real-world demonstrations have already shown that this risk is not only academic theory [1].
Facing this challenge, the paper at hand first introduces some potential automotive security attacks and some important automotive security threats. It then explains in more detail how to identify and evaluate potential security threats for automotive IT components based on theoretical security analyses and practical security testing. Lastly, we propose "automotive security evaluation assurance levels" (ASEAL) which define up to four discrete security testing levels.

1 Introduction

Suddenly car drivers all over the world witness spooky behavior of their Internet-enabled car infotainment units over the past few days. Out of the blue their navigation system jumps to another route, the unit calls the service on its own, or the display shows sculls and laughing, white masks. A quick analysis by security experts shows that the reason behind this behavior is a critical security breach of the GSM/LTE interface that enables unauthorized persons to access the software of the infotainment unit. But, how is it possible that this vulnerability had been missed, as the infotainment unit passed numerous tests? The answer is quite simple; even though there were several tests focusing on the functional safety, a systematic security evaluation containing theoretical analysis and practical tests had not been accomplished.

Luckily, this is only a potential scenario (yet) and not a real case but it clearly exposes a critical gap in automotive IT testing, which is not yet covered by existing functional testing procedures that are already well-established and conducted since several decades.

However, in contrast to functional testing, the systematic evaluation of automotive IT components regarding IT security is still in a very early stage. At the same time there is a strong need for security testing to be a part of the engineering procedure, not only due to scenarios like the introductive example, but also as a result of corresponding research activities [2], [3] and increasing demands made by public authorities [4]. Hence, this paper starts with a discussion about advantages and corresponding efforts of systematic security testing for the automotive industry. To do this we take into account recent security threats for automotive IT systems, ranging from odometer manipulation up to remote controlling of the steering system, and explain in detail why good security evaluations could have prevented most of the attacks.

The contribution of this paper is two-fold. First, we give an overview and a short introduction to the different aspects of embedded security evaluations such as theoretical security analyses, practical security testing, and verifiable security verification (cf. Figure **2**) especially regarding automotive onboard IT components.

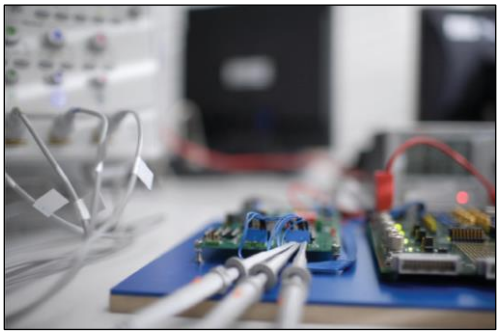


Figure 1: A practical testing setup for testing an embedded device at the authors' lab.

The second contribution of this work is a first proposal for establishing so-called "automotive security evaluation assurance levels" (ASEAL) which define up to four discrete security testing levels that determine which security analyses and tests should be part of a certain ASEAL and how "deeply" these security analyses and tests should be executed (cf. Section Table 1).

2 Automotive Security Threats

This section presents an overview of potential automotive security threats. To understand the type of threats, one must first have a basic understanding of the various automotive functionalities which can be targeted. Obviously there are direct security threats on driving safety such as manipulating the steering wheel or brakes, and indirect security threats such as distracting the driver by triggering odd vehicle behavior. Security researchers have successfully demonstrated that it is already possible to engage or disable the brakes or manipulate the steering wheel [3], [2], [1], [5] by maliciously injecting the relevant messages on the vehicle CAN bus. Indirect safety issues are

possible by injecting CAN messages to, for example, disable wipers or turning off the headlights when it is raining and dark outside.

Another type of security threat is targeting authoritative functionalities in the vehicle. For instance, the odometer logs the traveled distance, and, when selling a used car, it is attractive for an attacker to reduce the odometer value to increase the value of the car. In Germany, according to police investigations, around 2 million cars are subject to odometer manipulation per year with an average loss per vehicle of around 3000 € resulting in total losses of around 6 billion € per year [6]. Furthermore, critical data is stored in the ECUs such as crash data, data for insurances, or warranty indicators. Such data is also very attractive for malicious manipulations. For example, data such as vehicle speed, seat belt status, brake pedal position etc. are typically recorded in the seconds before a crash. A driver who has been involved in an accident could be motivated to change the recorded data to indicate that the brakes were applied when they really were not.

Moreover, since vehicles are becoming ubiquitously interconnected, increasing amounts of private data such as vehicle location, credentials to online services, or mobile payment data are stored in the vehicle as well. Attackers may be interested in stealing such data and misuse them directly or use as a stepping stone to launch further attacks, for instance, attacking an online service by stealing the respective credentials stored in the vehicle. Such private data could be extracted wirelessly by exploiting security vulnerabilities in services provided by communication interfaces such as Bluetooth and Wi-Fi, or through physical access to the OBD port, a USB port, or the ECU itself.

Another type of threat is theft of vehicles or valuable vehicle components such as airbags or head units, e.g. by abusing diagnostics commands to reprogram a new key. This functionality is typically used by workshop dealers when replacing a lost key, but can also be exploited by attackers to program a “thief key” for the vehicle that they are stealing [7]. There are other cases where attackers are able to send control messages to a vehicle to disable the alarm and unlock the doors, resulting in attackers being able to gain physical access to the interior of a vehicle [8].

3 Related Work

There is extensive work in IT automotive security testing conducted mostly by independent security researchers. For example, the Center for Automotive Embedded Systems Security (CAESS) [3], [2] has performed practical security evaluations of vehicles and found a number of issues. They have successfully exploited these vulnerabilities resulting in disabling of engine or brakes as well as in gaining remote access to internal systems. Both security testing of external interfaces to remotely access the vehicle as well as security testing of the in-vehicle network was performed. Other researchers [1] have shown practical attacks based on non-diagnostic CAN messages resulting in killing the engine, locking the brakes, or jerking the steering wheel while driving.

Lately, there have been various requests that urge OEMs to consider security and to perform more rigorous security testing. For example, Senator Edward Markey has sent a letter to 20 automotive manufactures inquiring about their view of and commitment to security [4]. Moreover, an independent group called *I am the cavalry* [9] has issued a 5 point security guide for automotive manufacturers in August 2014 urging them to follow this guide to improve automotive security. The five points are*: Safety by design, Third party collaboration, Evidence capture, Security updates,* and *Segmentation and isolation.*

Moreover, there exists an annual event [10] where OEMs provide vehicles for automotive security testing by independent researchers, engineers and students. The key idea is that the participating members at the event practically test the security of the vehicles by trying to find vulnerabilities. The OEMs typically have engineers participating in the event to understand what the specific vulnerabilities are as a short-term goal and to learn to consider security in the design (to get the mindset of an attacker when designing the solution) as a long-term goal.

Other industries also make use of security testing. For example, for ICS (industrial control systems), there is an EDSA (embedded device security assessment) certification [11] that includes practical security testing in terms of fuzz testing for the communication robustness testing component. For the banking industry, there is EMVCo certification [12] that includes a Security Evaluation process where general security performance characteristics and the suitability of use for smart card related products and IC (integrated circuits) chip-based tokens are evaluated. The purpose of the EMVCo evaluation is to assess whether the security features provided by the chip product are appropriately implemented. Furthermore, practical testing including penetration testing examines the interaction between the chip, operating system, and application to evaluate whether sensitive and secret information, as well as payment assets are adequately protected by the final chip product.

For the computer security certification, there is an international standard (ISO/IEC 15408 [13]) called Common Criteria for Information Technology Security Evaluation (CC). To achieve a particular Evaluation Assurance Level (EAL), the computer system must meet specific assurance requirements. These requirements typically involve design documentation, design analysis, functional testing and penetration testing.

In the US, requirements for government security are regulated by Federal Information Processing Standards (FIPS) [14]. The purpose for FIPS validation is that it assures users that a given technology has passed rigorous testing under either the Cryptographic Algorithm Validation Program (CAVP) or Cryptographic Module Validation Program (CMVP) by an accredited third-party lab and can be used to secure sensitive information. There exist several FIPSs, for example FIPS 140-2 [15] provides security requirements for cryptographic modules.

4 Embedded Security Evaluation

As discussed in the Section 2, modern vehicles can be open to various security risks. By applying in-depth security evaluations for an automotive IT system, for instance an

ECU, potential security weaknesses can be identified and countered before an attacker can exploit this weakness in the field and cause real financial or even safety damages. The earlier such a security evaluation is done within the developer cycle the less costly and time-consuming it is as well as security weaknesses can be found early and be closed effectively.

Automotive security evaluations can be done in theory as well as in practice. Theoretical security evaluation can (and should) be done during virtually all steps of the automotive development cycle, ideally already from the very beginning, when only a description of the vehicular IT system is available. Subsequent security evaluations for the next product development iterations can then be done very efficiently based on the results of the previous evaluation. In fact, the need for theoretical and practical security evaluations does not even end with series production of the corresponding IT system. Even in the field the IT component might need a security re-evaluation (and eventually also new countermeasures) due to ongoing development of new attacks or new results from security research. Practical security testing, of course, can only be conducted on an implementation of the target system, for instance with a first prototype. It is important to note that security evaluation can support but cannot replace mandatory security protection measures such as security by design or security engineering. In the following section we will explain the different approaches of security evaluation in embedded systems, see Figure **2**. For each of the three main categories we will shortly address the different sub-categories and explain the benefit of each method.

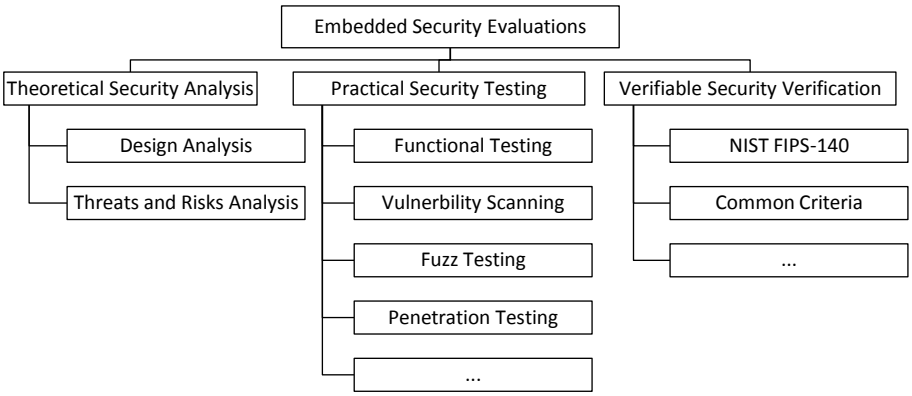


Figure 2: Overview of Embedded Security Evaluation Categories.

4.1 Theoretical Automotive Security Analyses

Theoretical security analyses are becoming gradually more common in the automotive context [16] and are applied to identify and understand the security weaknesses of an automotive IT system based on a paper-based evaluation of the corresponding system specifications and documentations. Depending on the level of scrutiny and the documents available, we differentiate between a more high-level *design analysis* and a more in-depth *threat and risk analysis*.

To conduct a **Design Analysis** of an automotive system only a theoretical description of the system is needed. Depending on the level of detail of these descriptions, e.g. high-level protocol descriptions up to explicit specifications, the depth and accuracy of the analysis varies. What is the goal of such a design analysis? First, the analysis can identify systematic flaws in the system even in an early state in the development since high level descriptions can be adequate for a design analysis. Secondly, the results can establish trust in the soundness of the system's architecture. To achieve these goals, the documents are inspected for potential attack points, e.g. weak cryptographic algorithms or possible attacks due to bad interaction of different standard protocols.

To categorize the identified vulnerabilities further and to detect the important flaws of the system that needs to be fixed, a **Threat and Risk Analysis** can be applied to the system. What are the important steps of such a threat analysis? Starting from the available documents, the system is analyzed and possible attacks identified; this step is similar to the design analysis. Additionally the difficulty of the corresponding attack procedure is rated for each of the attacks identified. The rating takes amongst others the required time, the needed expertise of the attacker, the equipment, and the needed access level into account. Furthermore, the potential damage of a successful attack is estimated in terms of safety, operational, and financial damages for the customer. Both values, the attack difficulty and the potential attack damage, result in an overall risk for a certain attack. Security vulnerabilities that result in attacks with a high risk are then critical candidates that should be fixed first.

Nonetheless, theoretical security analyses cannot find any implementation flaws or deviations of the implementation from the specification. Nor can a security analysis find vulnerabilities that are part of insufficiently documented specifications or flaws hidden in supplied components from third parties. To guard the system against such implementation issues, secure software development measures should be applied to the whole vehicular development process [17], [18]. But especially practical security testing, as described in the following section, can be used to identify possible vulnerabilities and cover the gap.

4.2 Practical Automotive Security Testing

Practical security testing can find implementation errors that could be exploited by an outside attacker, but also unspecified functionality and discrepancies to the specifications. Therefore, a thorough practical security test helps to establish trust in the soundness of the implementation. Furthermore, practical security tests help to estimate the actual difficulty of an attack against the target system. In general, practical security testing consists of at least four different steps (Figure 2) as described in the following paragraph.

In the first step, **functional security testing,** tests all security-related functions inside the test system for correct behavior and robustness. This step is similar to general functional testing but with focus on security functionality. A careful execution of this test can find implementation errors, discrepancies to the specification, and especially unspecified functionality that all might result in a potential security weakness. The next step, **vulnerability scanning,** tests the system for already known common security

vulnerabilities, for instance, known security exploits or (security) configurations with known weaknesses. **Fuzzing** goes even further and tries to find new vulnerabilities of an implementation by sending systematically malformed input to the target system to check for unknown, potentially security-critical system behavior. To test the security of the whole system, that means software and hardware, highly individual **penetration tests** can be applied in a last step. During a penetration test a "smart human tester" tries to exploit all the vulnerabilities which were found in the earlier steps in a "sophisticated way" based on many years of "hacking experience" with the goal to change the behavior of the target system. All these methods are explained in more detail in the next Section 5.

However, practical security testing, especially fuzzing and penetration testing, cannot give any assertion on completeness. Depending on the time and resources it is possible to miss larger systematic flaws. Hence, practical security testing cannot replace theoretical security analyses and should always be complemented by a theoretical analysis to identify possible attack paths. Additionally, Secure Software Development should be applied to the whole development process to minimize the total attack surface early on.

4.3 Verifiable Automotive Security Certifications

As discussed in Section 3, Security Certification is not a new idea ([11], [12], [13], [14]); although, there is no direct certification standard for the automotive industry, an overall Security Certification Standard which assures certain levels of security for an automotive system would serve extremely useful. A proposal for a verifiable automotive Security Certifications can be found in Chapter 6.

To certify a system, first theoretical and practical analyses of the system are needed to understand the level of security and the risk of the system. The common idea is that for a certain certification level, a system has to pass a set of tests. In other words, a Security Certificate assures that the process of specification, implementation, and evaluation of the system at hand has been conducted at a standard and repeatable manner with a certain level of security. Therefore**, Security Certifications** help to compare the security level of different target systems and to build trust with customers.

5 Practical Automotive Security Testing

Corresponding publications from the last years, e.g. [16] and [19], focus on theoretical security evaluation without detailing the importance of practical security testing. Within this section, we close this gap and explain why the automotive industry can gain significantly especially from practical security testing.

As explained earlier, a good theoretical security evaluation can spot many issues and counter them; however, sometimes even a well-designed automotive system can suffer from substandard implementations, poor configurations, or physical weaknesses [20]. For example, the random seed for security access could be obtained from a hardware register but the register has no entropy at this stage of the booting process, resulting in a constant seed; hence, we stress the importance of practical security testing.

As illustrated in Figure **3**, there are various techniques to conduct a practical security evaluation. Depending on the time and effort one wants to spend, the range of these tests spans from simple interface scans for known vulnerabilities to invasive techniques such as micro-probing to recover secret data from an automotive component. Other methods, such as fuzzing to find unknown vulnerabilities of a component or power analysis to recover cryptographic keys lie between these extremes. A combination of all these approaches is a powerful method to find security vulnerabilities in automotive components that may have been missed in or have not been covered by theoretical security analyses.

For all the approaches the tester needs to have access to the actual software and hardware of the target system, cf. Figure **1**. Furthermore, for functional testing also the specification of the system is needed, and for all other methods supporting software, for instance restbus simulation, may be needed to run the hardware device. Moreover, special testing hardware and software is needed, for example, JTAG-debuggers or special signal generators.

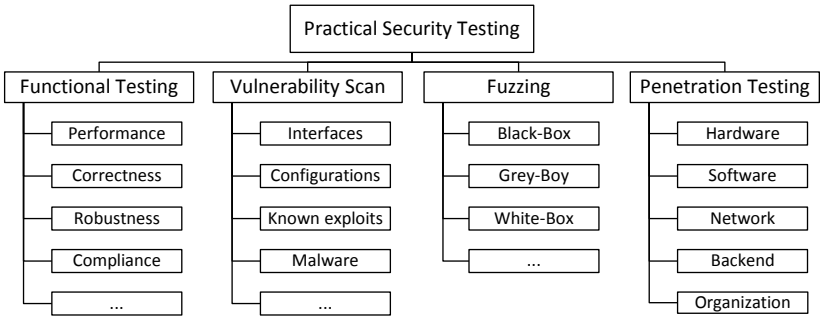


Figure 3 Classification of Practical Security Testing Methods.

5.1 Functional Automotive Security Testing

Functional automotive security testing ensures the general compliance to specifications and standards of the implemented security functionality, for instance, encryption algorithms and authentication protocols, of a vehicular IT system. However, the algorithms are not only tested for correct behavior according to the specification but also for robustness. Furthermore, performance of (often computationally intense) security algorithms is tested to identify potential bottlenecks that might affect the overall security performance. As a result, functional security testing ensures dependable security functionality and that a functional weakness does not create any exploitable security threats.

In many cases, standard implementations, such as OpenSSL [21] are not suitable for use in the automotive domain due to various constraints, and therefore a much wider spectrum of cryptographic and security relevant implementations are in use. Performance or size limitations need to be considered but also safety standards such as MISRA-C [22] must be fulfilled. Furthermore, a wide range of automotive specific security protocols are in use, such as secure flash algorithms or secure communication, secure OBD, theft protection, and upcoming vehicle-to-x (V2X) communication. It is

vital that those security implementations are subject to thorough functional security testing.

Functional security is usually achieved by testing the implementation against official test vectors (if available) or independent implementations. Many cryptographic algorithms could contain specific corner cases that could lead to security vulnerabilities, for example subtle flaws in numeric implementations that trigger only in one out of 4 billion random cases. These corner cases must be tested with specially constructed test vectors and by running lengthy tests. Furthermore, many modern cryptography schemes and security implementations rely on secure random number generators. In order to gain trust into the security of such a random number source, extensive statistical testing is required. Finally, in the highly performance- and cost-sensitive automotive environment, performance testing can help with correct dimensioning of hardware and enable an optimal choice of security algorithms and parameters.

5.2 Automotive Vulnerability Scans

Vulnerability scans are used to examine all relevant applications, source codes, networks, and backend infrastructures of an automotive system for known security weaknesses from a continuously updated database of known automotive security vulnerabilities.

There are numerous different variations of vulnerability scanning. Firstly, the code of the software/firmware running on the system can be scanned, identifying, for example, buffer overflows and heap overflows by using static and dynamic analyses. Depending on the respective analysis tools, this must be done on source level and binary level. Note that it is not necessarily clear that the result is the same in both cases. The compilation process may introduce more security vulnerabilities, for example by removing security checks during the optimization step, or through faulty compilers. Therefore, compiler settings must be examined closely.

Secondly, the system can be scanned for open ports and interfaces, and also for available services running on these interfaces. In automotive systems, this encompasses classical IT interfaces such as IP communication on Ethernet, Wi-Fi, or cellular internet. Scanning these interfaces is especially valuable due to the fact that a whole range of operating systems, network stacks, applications, and libraries are typically re-used, where a large base of vulnerabilities is already known and can be tested for automatically, as done in OpenVAS [23]. Scanning includes reconnaissance port scans, as well as deep scans of specific vulnerabilities. In addition, the automotive environment has special automotive bus systems such as CAN, which have no equivalent in classical IT, but which are highly standardized. This means that automatic scanning tools are well-suited to provide a first overview of vulnerabilities. In this context, scans of diagnostic functionality are notable, as those are likely to contain weakly documented security critical functionality, such as development or debugging functionality.

As a third form of vulnerability scanning, the configuration for the whole system can be analyzed to identify security gaps, e.g. access to critical functions possible without authentication. Automated scans can also test for the presence of different authentication

mechanisms securing the same critical functions. To conclude, vulnerability scanning ensures that a system is secure against known attacks, which can easily be tried out by attackers and therefore are very likely attacks.

5.3 Automotive Fuzzing

Fuzzing is a technique used for a long time to test software and IP networks by exposing the implementation to unexpected, invalid, or random input with the hope that the target will react in an unexpected way, and thereby, to discover new vulnerabilities. The reaction of the target can range from strange output over unspecified behavior up to crashes. Fuzzing as a testing technique for automotive target systems is relatively new; although, modern vehicles have many similarities to common computer networks. In fact, ECUs can be viewed as small computers, running different software, that are connected by different network types such as CAN, FlexRay, or MOST. Hence, it is quite natural to consider the idea to apply fuzz testing also to automotive target systems as part of the security testing process.

In general, fuzzing consists of three different steps: firstly the creation of the input for the target, secondly the delivery of the input to the target and lastly the monitoring of the target system to detect errors in the program flow. Since fuzzing is widely used in the computer world, fuzzing tools such as Peach [24] already exists. Peach has a powerful fuzz generator that can be adapted individually for different protocols such as UDS. The input generated by the fuzz generator is then delivered to the target using the required transport protocol. The target system is monitored to detect possible vulnerabilities. The monitoring process can range from inspection of return values up to the usage of debuggers which observe the internal status of the target device. In the end, all identified unusual behavior has to be analyzed by an expert to detect exploitable vulnerabilities. Examples of such exploitable bugs are insufficient input validation or undocumented functionality, e.g. open debug or configuration interfaces. One famous example for insufficient input validation is the Heartbleed Bug [25] of OpenSSL, which allows reading out critical data since length parameters are not double-checked.

In the automotive context, fuzzing can be applied to diagnostics protocols, such as UDS, or to automotive network protocols, e.g. CAN, FlexRay, MOST or LIN. However, classical fuzzing targets, i.e. IP based networks, play an increasing role in modern vehicles. Hence, automotive security testing also benefits from experiences made in fuzz testing of classical protocols and modern software applications, e.g. cellphone apps.

5.4 Automotive Penetration Testing

Automotive penetration tests are motivated by either IP protection or authoritative functionalities that rely on the integrity of the target system against interests of physically present persons. Examples are theft protection, component protection, odometer manipulations, feature activation, protection from false warranty claims from “tuned” vehicles, or safety functionality. However, in the modern connected world remote attacks begin to be a real threat [26] and this motivates penetration tests of all

communication channels, the backend and relevant organizational processes (e.g. for social engineering attacks).

Typically, penetration tests of a physical device start with general reconnaissance, which includes enumerating interfaces, determining components and their connections on the PCB, gathering specifications available to a hypothetical attacker, and in general, any information that can be helpful in further attacks. Using the information acquired in the first step, further attacks can be planned. A second step may include attacks of local external interfaces such as USB, serial ports, or attacks of the hardware itself. To attack the hardware, usually the tester tries to find overlooked or undocumented debug access interfaces, or gain access to ECU-internal interfaces such as memory buses. More advanced methods require etching open chip packages and accessing the actual silicon chip. In a third step all communication channels to the device, such as the CAN bus, Ethernet, or Wi-Fi, are analyzed and used to are used to attack the target device. Depending on target system and the scope of the penetration test, further attacks against the backend can be conducted. Regarding penetration testing, there are three specific forms: black-box tests, white-box tests, and grey-box tests. In the following we will describe the approaches in more detail.

For black-box testing, the tester is provided with practically no documentation or specifications, except information that could also be acquired by a real world attacker. The advantage of this method is that this results in a very realistic simulation of a real attack. As a disadvantage, the penetration tester must spend a lot of time on basic reverse engineering, and there is a good chance that deeper attack paths are not discovered because the tester did not circumvent easier first-line defense mechanisms, whereas a real attacker may later break such defense mechanisms, by luck, because more information has become public as a result of the state-of-the-art advancing, or because he invested more resources than the tester.

For a white-box test, the tester is provided with full specifications and documentation for the device under test. This means he is able to specifically target weaknesses, and has more resources available, which he did not have to spend on gaining information. Both reasons improve the efficiency of the test. The disadvantage of white-box testing is that the conditions are not nearly as realistic as in black-box tests, giving a less reliable estimation of attack difficulty and likelihood.

Grey-box-tests represent a middle ground between black-box and white-box testing. For a grey-box test, the tester receives partial information, concerning a specific sub-system that is in focus or information that a specific attacker such as an insider could have acquired. A step-wise approach is also possible, where the tester receives more information or access after having shown the basic presence and exploitability of a vulnerability without having to fully develop the attack itself. This optimizes the ratio of test efficiency and realism.

However, penetration attacks are not necessary limited to attacks on the devices hardware, software and networks, but can also include attacks on the organizational implementation like social engineering attacks or weak organizational processes.

6 Automotive Security Assurance Levels

In this chapter we give a first proposal of so-called "automotive security evaluation assurance levels" (ASEAL) which define up to four discrete security testing levels that determine (i) the size of security evaluation scope, that means which security analyses and tests have to be executed for a certain ASEAL and (ii) how "deeply" and thoroughly these security analyses and tests have to be executed. The goal for ASEAL is to make security evaluations comparable and, in consequence, to make it possible to assign standardized levels of minimum security assurance to each automotive onboard IT component. Our ASEAL approach hence combines and extends the rather specific vehicle-to-vehicle communication security evaluation approach using "Trust Assurances Levels" [19] with the rather general ideas from overall security evaluation framework of Common Criteria [13] to form a standardized automotive IT security assurance level.

Concretely, we define – depending on the required ASEAL A, B, C or D – minimum requirements regarding evaluation scope and depth for each theoretical security analysis and each practical security test. Table 1 gives a first (yet incomplete) overview about the idea in general by using exemplary classifications and potential security evaluation assurance levels.

Table 1: Exemplary Automotive Security Evaluation Assurance Levels (ASEAL).

Evaluation Category	Type of Automotive Security Evaluation		Scope and Depth of Automotive Security Evaluation for each ASEAL			
			ASEAL A	ASEAL B	ASEAL C	ASEAL D
Theoretical Security Analyses	TRA: Security Threats and Risks Analysis		TRA1	TRA2	TRA3	TRA4
	SDA: Security Design Analysis		SDA1	SDA2	SDA3	SDA4
	DEV: Security Development Analysis		--	--	DEV1	DEV2
	DEP: Security Deployment & Processes Analysis		--	DEP1	DEP2	DEP3
Practical Security Testing	FST: Functional Security Testing		FST1	FST2	FST3	FST4
	VUL: Vulnerability Scanning		--	VUL1	VUL2	VUL3
	SYF: Systematic Fuzzing			SYF1	SYF2	SYF3
	Penetration Testing	LPA: Logical Penetration Attacks	--	--	LPA1	LPA2
		IPA: Invasive Penetration Attacks	--	--	IPA1	IPA2
		ORA: Organizational Pen. Attacks	--	--	ORA1	ORA2

In the following we present four exemplary definitions (i.e. two theoretical analyses and two practical security tests) for the proposed security evaluation types including some first proposals for the ASEAL-depending evaluation scope and depth.

Table 2: Exemplary ASEAL definitions for automotive security threats & risks analyses

Type	TRA: Security Threats and Risks Evaluation	
Description	Analyzing potential automotive attackers (i.e. owner, garage, competitor, third party), attack scenarios (e.g. undermined business models, parameter manipulation, IP theft, sabotage), and attack paths (e.g. telematics interface, NFC, OBD, Internet).	
For ASEAL A	TRA1	Informal analyses with standard automotive attacker model and attack paths.
For ASEAL B	TRA2	TRA1 + complete attack tree including weighted attack paths for all known automotive security attacks.

For ASEAL C	TRA3	TRA1/2 + methodically tested and verified (cf. CC EAL 4 [13]) including thorough "Darknet" investigations and research for weighting of potentially yet unknown attack paths.
For ASEAL D	TRA4	TRA1/2/3 + semi-formal tested and verified (cf. CC EAL 5/6 [13]).

Table 3: Exemplary ASEAL definitions for security deployment & process analyses

Type	DEP: Security Deployment & Process Evaluation	
Description	Analyzing the security of the component integration, deployment and all other security-related processes at the backend (e.g. key creation/distribution, web interface, access authorizations, and backend security parameters), automotive component (e.g. key injection, access control, initialization, personalization), and communications (e.g. key length, algorithm, key exchange) during production, operation, and phase-out.	
ASEAL A	--	No evaluation required.
ASEAL B	DEP1	Analyses initial setup and initial security configuration.
ASEAL C	DEP2	DEP1 + production, maintenance, and change deployment & processes.
ASEAL D	DEP3	DEP1/2 + phase-out and deactivations processes.

Table 4: Exemplary ASEAL definitions for systematic automotive fuzzing tests

Type	SYF: Systematic Automotive Fuzzing	
Description	Analyzing the security of the target by systematic fuzzing of software communication stacks (e.g. CAN stack, Ethernet stack) and external interfaces (e.g. USB)	
ASEAL A	--	No fuzzing tests required.
ASEAL B	SYF1	Fuzzing of all standard communication protocols (e.g. UDS, TCP/IP) and external interfaces (e.g. Wi-Fi, Bluetooth)
ASEAL C	SYF1	SYF1 + fuzzing of non-standard communication protocols (e.g. special RS232 protocols, proprietary CAN protocols)
ASEAL D	SYF2	SYF2 + extended fuzzing method (i.e. evolutionary fuzzing) for all communication protocols and interfaces

Table 5: Exemplary ASEAL definitions for physical automotive attack tests

Type	IPA: Invasive Penetration Attacks	
Description	Analyzing the attack probability and costs for all kind of attacks (including insider, offline, side-channel, or fault injection attacks) which are very critical for most embedded devices working within a "hostile environment" where attackers have full physical control about the device itself and its environment (e.g. power supply, input signals, temperature etc.).	
ASEAL A	--	No invasive/physical attacking tests required.
ASEAL B	--	No invasive/physical attacking tests required.
ASEAL C	IPA1	Simple attacks such JTAG attacks, memory dumps, simple I/O manipulations, offline attacks, or insider attacks (i.e. w/ user/owner privileges).
ASEAL D	IPA2	SYF1 + extended physical and invasive attacks such as side-channel attacks, fault injections, and complex manipulations (e.g. connection cutting, re-wiring, re-fuse), and readouts (e.g. micro-probing, optical memory read-out).

The above tables of course are only first high-level descriptions to demonstrate the general idea. A final ASEAL approach however would (in contrast to Common Criteria) (i) provide very concrete details and security requirements from typical automotive security risk cases and (ii) restrict minimum security requirements to protect against all *reasonable* automotive security attacks, but without trying to protect against all *feasible* automotive security attacks for instance by intelligence services or cyber war actors with virtually unlimited resources.

7 Conclusion & Open Challenges

This paper shows the strong need for systematic automotive security evaluation and discusses different methods for theoretical and practical security evaluation of automotive IT components. We focused especially on practical security testing for automotive components such as automotive penetration testing, since practical security testing is still relatively new to the automotive domain.

We believe theoretical and practical security evaluations will become standardized and mandatory procedures, similar to safety testing in ISO 26262 [27], to fulfill state-of-the-art product liability requirements and to protect various upcoming automotive business models (e.g. on pay per use basis). This of course requires serious efforts from OEMs, suppliers and security experts to establish necessary automotive security testing expertise, testing standards and testing infrastructures.

Moreover, we propose a standardized security evaluation process which provides four different assurance levels (ASEAL). Using ASEAL it is possible to assure that a specific automotive IT system has been security tested to a certain defined level. This can help to ensure worldwide comparable minimum security protection levels depending on the respective security risks similar to today's automotive safety integrity levels (ASIL).

References

[1] C. Miller und C. Valasek, „Adventures in Automotive Networks and Control Units," DEFCON 21 Hacking Conference, 2013.

[2] S. Checkoway, D. McCoy, B. Kantor, D. Anderson, H. Shacham, S. Savage, K. Koscher, A. Czeskis, F. Roesner and T. Kohno, "Comprehensive Experimental Analyses of Automotive Attack Surfaces," in *USENIX Security*, San Francisco, CA, USA, 2011.

[3] K. Koscher, A. Czeskis, F. Roesner, S. Patel, T. Kohno, S. Checkoway, D. McCoy, B. Kantor, D. Anderson und H. a. o. Shacham, „Experimental security analysis of a modern automobile," *Security and Privacy (SP), 2010 IEEE Symposium on*, 2010.

[4] E. Markey, "As Wireless Technology Becomes Standard, Markey Queries Car Companies about Security, Privacy," Press Release of the US Senator for Massachusetts, Massachusetts, USA, 23.12.2013.

[5] C. Miller und C. Valasek, „A Survey of Remote Automotive Attack Surface," 2014.

[6] A. V. Thiemel, M. Janke und B. Steurich, „Speedometer Manipulation – Putting a Stop to Fraud," *ATZ elektronik worldwide Edition*, 2013-02.

[7] EXTREMETECH, „Hack the diagnostics connector, steal yourself a BMW in 3 minutes," [Online]. Available: http://www.extremetech.com/extreme/132526-hack-the-diagnostics-

connector-steal-yourself-a-bmw-in-3-minutes]. .

[8] EXTREMTECH, „Hackers can unlock cars via SMS,“ [Online]. Available: http://www.extremetech.com/extreme/91306-hackers-can-unlock-cars-and-meddle-with-traffic-control-systems-via-sms.

[9] The Cavalry, 2014. [Online]. Available: https://www.iamthecavalry.org/.

[10] Battelle , „Annual Battelle Cyberauto Challenge,“ [Online]. Available: http://www.battelle.org/site/cyber-auto-challenge.

[11] ISASecure, „Embeded Device Security Assurance (EDSA),“ [Online]. Available: http://www.isasecure.org/ISASecure-Program/EDSA-Certification.aspx.

[12] EMVCO, „EMVCO,“ [Online]. Available: http://www.emvco.com/.

[13] International Organization for Standardization (ISO), „ISO/IEC 15408-3: Information technology -- Security techniques -- Evaluation criteria for IT security,“ 2008.

[14] certicom, „FIPS Validation: what is it?,“ https://www.certicom.com/index.php/fips-validation-what-is-it.

[15] National Institue of Standards and Technology (NIST), „"FIPS PUB 140-2: Security Requirements for Cryptographic Modules,“ 2007.

[16] M. Wolf and M. Scheibel, "A Systematic Approach to a Quantified Security Risk Analysis for Vehicular IT Systems," in *Automotive - Safety & Security*, Karlsruhe, 2012.

[17] CERT, „Secure Coding Standards,“ [Online]. Available: http://www.cert.org/secure-coding/research/secure-coding-standards.cfm?.

[18] SAFECode, „Fundamental Practices for Secure Software Development,“ 2011.

[19] D.Angermeier, A.Kiening, H.Seudié, T.Stodardt und M.Wolf, „Trust Assurance Levels of Cybercars in V2X Communication,“ in *Workshop on Security, Privacy, and Dependability for CyberVehicles (CyCAR 2013) co-located with ACM CCS*, Berlin, 2013.

[20] D. K. Oka, C. Vuillaume and T. Furue, "Vehicle ECU Hacking," in *ASIACCS*, Kyoto, Japan, 2014.

[21] The OpenSSL Project, [Online]. Available: https://www.openssl.org/.

[22] M. I. S. R. Association und M. I. S. R. A. Staff, MISRA C:2012: Guidelines for the Use of the C Language in Critical Systems, Motor Industry Research Association, 2013.

[23] „OpenVAS - Open Vulnerability Assessment System,“ [Online]. Available: http://www.openvas.org.

[24] DEJA VU SECURITY, „Peach Fuzzer platform,“ 2014. [Online]. Available: http://peachfuzzer.com/products.

[25] „The Heartbleed Bug,“ 2014. [Online]. Available: http://heartbleed.com/.

[26] heise, *BMW ConnectedDrive gehackt*, heise.de, 2015.

[27] International Organization for Standardization (ISO), „ISO 26262: Road vehicles - Functional safety,“ 2011.

[28] M. Wolf, Security Engineering for Vehicular IT Systems - Improving Trustworthiness and Dependability of Automotive IT Applications, Vieweg+Teubner Verlag, 2009.

[29] Alliance of Automobile Manufacturers, „Auto Cyber-Security: Continual Testing, Checks and Balances,“ July 2014. [Online]. Available: http://www.autoalliance.org/auto-innovation/cyber-security.