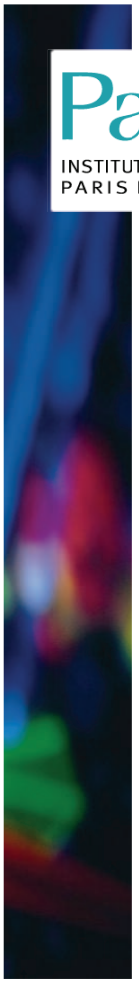
Security and privacy in automotive on-board networks

Hendrik Schweppe



ParisTech
INSTITUT DES SCIENCES ET TECHNOLOGIES
PARIS INSTITUT OF TECHNOLOGY

TELECOM ParisTech

2012-ENST-062

EDITE ED 130

Doctorat ParisTech

T H È S E

pour obtenir le grade de docteur délivré par

Télécom ParisTech

présentée et soutenue publiquement par

Hendrik SCHWEPPE

le 8 novembre 2012

Sécurité et protection de la vie privée dans les systèmes embarqués automobiles

Directeur de thèse : **Yves ROUDIER**

Jury

M. Erland JONSSON, Professeur, Chalmers, Göteborg, Suède	Rapporteur
M. Jean-Marc ROBERT, Professeur, École de Technologie Supérieure, Québec, Canada	Rapporteur
M. Joaquin GARCIA-ALFARO, Maître de Conférences, Télécom SudParis, France	Examinateur
M. Renaud PACALET, Directeur d'Études, Télécom ParisTech, Sophia-Antipolis, France	Examinateur
M. Panagiotis PAPADIMITRATOS, Professeur, KTH, Stockholm, Suède	Examinateur
M. Yves ROUDIER, Maître de Conférences, EURECOM, Sophia-Antipolis, France	Examinateur
M. Benjamin WEYL, Docteur-Ingenieur, BMW Group, München, Allemagne	Invité

T H È S E

Télécom ParisTech
Ecole de l'Institut Télécom – membre de ParisTech
46, rue Barrault – 75634 Paris Cedex 13 – Tél. + 33 (0)1 45 81 77 77 – www.telecom-paristech.fr

Security and Privacy in Automotive On-Board Networks

Thesis

Hendrik C. Schweppe
hendrik.schweppe@eurecom.fr

École Doctorale Informatique, Télécommunication et Électronique, Paris
ED 130

November 8, 2012

Advisor:

Prof. Dr. Yves Roudier
EURECOM, Sophia-Antipolis

Reviewers:

Prof. Dr. Erland Jonsson,
Chalmers, Gothenburg

Prof. Dr. Jean-Marc Robert,
ETS, Québec

Examiners:

Prof. Dr. Joaquin Garcia-Alfaro,
Télécom SudParis

Prof. Renaud Pacalet,
Télécom ParisTech, Sophia-Antipolis

Prof. Dr. Panagiotis Papadimitratos,
KTH, Stockholm

Invitee:

Dr.-Ing. Benjamin Weyl,
BMW AG, Munich

To Annette
and my parents.

Acknowledgements

I would like to express my gratitude to the people who have helped me to pursue my Ph.D. studies. First and foremost, this is Prof. Dr. Yves Roudier, my *directeur de thèse*, who always supported me with ideas, fruitful discussions and trusted me to oversee a large part of the research done in the EVITA project. My peer at BMW Forschung und Technik, Dr.-Ing. Benjamin Weyl is equally important: He was the driving force behind the project, and at the same time gave me many inspirations and ideas for the concepts of this thesis. Benjamin made it possible for me to visit BMW in the scope of the project, and to get to know the industrial side of research. Thank you. I also would like to thank the jury members and especially the reviewers of my thesis, Prof. Dr. Erland Jonsson and Prof. Dr. Jean-Marc Robert. The early feedback I received from Prof. Jonsson was very detailed and helpful. Furthermore I express my appreciation to my colleagues and friends at the EURECOM institute. My deepest gratitude belongs to my family and Annette, who continuously supported me at every hour of the day.

Abstract

In recent decades, vehicles have been equipped with an increasing number of electronic features and controllers. They have become a vital part of automotive architecture. This architecture consists of an internal network of microcontrollers and small computers, called Electronic Control Units (ECUs). Such ECUs may be part of an entertainment system, which will interact with the driver, or they complement technical and mechanical systems such as power steering, brakes, or engine control. Every ECU is usually connected to one or more networks as well as a number of sensors and actuators.

Vehicles have become multi-connected places: i) Entertainment systems allow data to be retrieved directly from the internet, typically traffic conditions, weather or navigational information, ii) Increasingly consumer devices are being connected by wired and wireless interfaces in order to control vehicle functions or distribute multimedia content, iii) Assistance functions to augment traffic safety and efficiency are currently being standardized, allowing vehicles and infrastructure units to communicate autonomously via dedicated radio channels.

All of these new communication interfaces should be properly secured, as failure to do so could have severe consequences, such as loss of control over the vehicle or private data being accessed by third party applications, which could, for example, record conversations or track usage behavior. Recent security analyses show that current vehicle architectures are vulnerable to the above described threats. It has been shown that by exploiting implementation flaws, attackers can control the vehicle's behavior from a device inside the car or even remotely.

This dissertation focuses on securing in-vehicle networks. Historically, vehicle buses such as the Controller Area Network (CAN) were considered as isolated embedded systems. However, an effective isolation of on-board networks is difficult if not impossible to achieve with the rises of connectivity inside the vehicle for internal functions and, at the same time, for third party devices and internet services. Upcoming safety and assistance functions use Car-to-Car and Car-to-Infrastructure communication (Car2X). These assistance functions rely on remotely received data. It is imperative that these data are trustworthy. A high level of trust can only be achieved by securing the on-board platform as a whole, and by protecting both the integrity and the authenticity of network communication as well as software execution.

The main contributions of this thesis are i) an approach to securing the communication of in-vehicle networks, ii) an approach to applying dynamic data flow analysis to the distributed embedded applications of on-board networks, by using taint-tag tracking in order to detect and avoid malicious activities, iii) working prototypes for different aspects of the overall security problem, showing simulations and real-world results of the techniques developed in this thesis.

The approach that is presented combines multiple mechanisms at different layers of the vehicular communication and execution platform. Cryptographic communication protocols are designed and implemented in order to authenticate data exchanged on the buses. Hardware Security Modules (HSMs) are used to complement the actual microcontroller by providing a secure storage and by acting as a local root of trust. We distribute usage-restricted symmetric key material between HSMs. Their use is limited to certain functions, like generating or verifying authentication codes. Thereby, they can be used asymmetrically for group-communication patterns. This is a common communication paradigm in automotive applications, in particular for distributing vehicle-wide signals. A proof of concept system has been implemented as part of this thesis, showing the feasibility of integrating security features on top of automotive buses and for use with Car2X communication. We simulated the behavior of a CAN network and compare our results for different network designs with data collected from a real vehicle and with simulations based on a Simulink toolkit.

In order to account for untrusted program code, we use a distributed data flow tracking based approach for securing code execution on the ECUs of the automotive network. This means that a high level of trust can be placed in applications even when mechanisms, such as software review and applications signatures, fall short of the desired security levels, or cannot be applied at all. If this approach is taken then the use of applications of unknown origin along side those controlling critical functions becomes possible. In addition to plain policy rules, we use a declarative approach to represent the kind of data used on communication links. Binary instrumentation techniques are used to track data flows throughout the execution and between control units.

For the Car2Car Communication Consortium Forum in November 2011, a part of the prototype implementations was integrated into two research vehicles to demonstrate an "Active Brake" safety scenario using secure in-vehicle and Car2X communication. It demonstrated the effectiveness and applicability of our communication security solution.

Zusammenfassung

Die Entwicklungen und Neuerungen rund um das Automobil ist in den letzten Jahren in zunehmendem Maße von elektronischen Funktionen und einer Ausweitung der Kommunikationsschnittstellen geprägt. Durch die Zunahme an Schnittstellen ist das Automobil zu einem Knotenpunkt der Vernetzung geworden. Applikationen verarbeiten interne Fahrzeugdaten, Daten aus dem Internet und von Mobiltelefonen, sowie in Zukunft auch von anderen Fahrzeugen mithilfe der sogenannten Car2X Kommunikation.

Durch Abhängigkeiten können die verschiedenen Systeme nicht komplett isoliert werden können. Daher müssen die untereinander verbundenen Bussysteme und Computerplattformen entsprechend abgesichert werden. Der erste Teil dieser Dissertation beschäftigt sich mit Techniken, die den sicheren Austausch von Daten im Fahrzeugnetz, speziell dem CAN Bus, ermöglichen. Hierzu wird ein Protokoll zur sicheren Verteilung symmetrischer Sitzungsschlüssel zwischen Steuergeräten vorgestellt, das auf sogenannte Hardware Security Module (HSM) aufbaut. Das HSM erlaubt der Schlüsselverteilung eine asymmetrische Nutzung von symmetrischem Schlüsselmaterial in 1:n Kommunikationsgruppen, wie sie im Fahrzeug typischerweise verwendet werden.

Der zweite Teil beschäftigt sich mit der Erkennung von Angriffen und einem Ansatz zur Nachverfolgbarkeit von Informationsflüssen. Diese DIFT genannte Technik zur Verfolgung von Informationen während Ausführung und Kommunikation wird für zweierlei Zwecke benutzt. Im Sinne klassischer Intrusion Detection können Datenströme hinsichtlich der verwendeten Daten analysiert werden, wodurch eine Erkennung von fremd eingebrachten Daten (z.B. durch Softwareschwachstellen) ermöglicht wird. Weiterhin wird das Verfahren eingesetzt, um nachladbare Applikationen, wie sie im Automobil bereits Einzug halten, bei der Nutzung von Daten zu kontrollieren, so dass private Daten nur für zulässigen Kommunikationswege eingesetzt werden können.

Beide Ansätze sind in einer Middleware umgesetzt worden und werden anhand dreier fahrzeugspezifischer Szenarien demonstriert und evaluiert. Hierbei wird besonders auf die Anwendbarkeit im automobilen Umfeld geachtet.

Im Rahmen des EU Projekts EVITA wurden Ergebnisse dieser Dissertation in zwei Fahrzeugdemonstratoren auf dem Car2Car Communication Forum 2011 praktisch umgesetzt.

Résumé

L'informatique de bord est maintenant devenue partie intégrante de l'architecture réseau des véhicules. Elle s'appuie sur l'interconnexion de microcontroleurs appelés Electronic Control Unit (ECU) par des bus divers. On commence maintenant à connecter ces ECUs au monde extérieur, comme le montrent les systèmes de navigation, de divertissement, ou de communication mobile embarqués, et les fonctionnalité Car2X. Des analyses récentes ont montré de graves vulnérabilités des ECUs et protocoles employés qui permettent à un attaquant de prendre le contrôle du véhicule.

Comme les systèmes critiques du véhicule ne peuvent plus être complètement isolés, nous proposons une nouvelle approche pour sécuriser l'informatique embarquée combinant des mécanismes à différents niveaux de la pile protocolaire comme des environnements d'exécution. Nous décrivons nos protocoles sécurisés qui s'appuient sur une cryptographie efficace et intégrée au paradigme de communication dominant dans l'automobile et sur des modules de sécurité matériels fournissant un stockage sécurisé et un noyau de confiance. Nous décrivons aussi comment surveiller les flux d'information distribués dans le véhicule pour assurer une exécution conforme à la politique de sécurité des communications. L'instrumentation binaire du code, nécessaire pour l'industrialisation, est utilisée pour réaliser cette surveillance durant l'exécution (par data tainting) et entre ECUs (dans l'intergiciel).

Nous évaluons la faisabilité de nos mécanismes pour sécuriser la communication sur le bus CAN aujourd'hui omniprésent dans les véhicules. Une preuve de concept montre aussi la faisabilité d'intégrer des mécanismes de sécurité dans des véhicules réels.

Contents

Abstract v

1 Introduction 1

1.1 Vehicle Electronics . . . 1
1.2 Motivation . . . 2
1.3 Problem Description . . . 4
1.4 Goals . . . 5
1.5 Structure of the Thesis . . . 8

2 State of the Art 9

2.1 Automotive Security . . . 10
2.1.1 Vehicle Components and Vulnerabilities . . . 11
2.1.2 Vehicle Security Concepts . . . 14
2.1.3 Ongoing Research . . . 15
2.1.4 Current Practice . . . 16
2.2 Software Security . . . 20
2.2.1 Overview and Classification of approaches . . . 21
2.2.2 Techniques and Methods for Information Flow Tracking . 26
2.3 Dynamic Application Environments and Platform Security . . . 29
2.3.1 Application Environments . . . 29
2.3.2 Example for Closed Application Environments: An App Store . . . 31
2.3.3 Attacks on iOS Security . . . 34
2.3.4 Complexity and Cost of Approaches . . . 35
2.4 Conclusion . . . 36

3 Environment **39**

3.1 Scenarios . . . 39

3.1.1 Scenario I: Active Brake (Car2X) . . . 39

3.1.2 Scenario II: Playing Music . . . 43

3.1.3 Scenario III: Driver Adaptation . . . 43

3.2 Attacker Model . . . 44

3.3 Possible Attack Vectors . . . 46

3.3.1 Attacks: Network Communication . . . 47

3.3.2 Attacks: Host Based Intrusions . . . 48

3.4 Towards A Secure In-Car Architecture . . . 48

3.4.1 Software Security: The Framework . . . 49

3.4.2 Communication . . . 49

3.4.3 Policy Decision . . . 50

3.5 Hardware: The Hardware Security Modules . . . 51

3.5.1 Key Usage . . . 51

3.5.2 The HSM programming interface . . . 51

3.5.3 Prototype Platform . . . 52

3.5.4 Performance . . . 52

3.5.5 Comparison with other Secure Hardware . . . 53

4 Communication Security **55**

4.1 Key Distribution in Embedded Environments . . . 56

4.1.1 Asymmetric Usage of Symmetric Keys . . . 56

4.1.2 Dynamic Key Exchanges . . . 57

4.1.3 Multi-Criteria Setting of Secure Communication . . . 58

4.1.4 The Protocol . . . 58

4.1.5 Multi-Domain deployment . . . 60

4.1.6 Initial Key Distribution . . . 64

4.1.7 Maintenance and Part Replacement . . . 64

4.2 Securing CAN Bus Communication . . . 67

4.2.1 Technical Background . . . 67

4.2.2 CAN Transport Protocol . . . 68
4.2.3 Truncation of Cryptographic Authentication Codes . . . 69
4.2.4 Implications for CAN bus communication . . . 71
4.3 Related Work . . . 72

5 Dynamic Platform Security 75
5.1 Intrusion Detection and Response . . . 77
5.1.1 Architecture: Vehicular Network and Host Based Intrusion Detection System . . . 78
5.2 Distributed and Dynamic Information Flow . . . 81
5.2.1 Data Flow Tracking . . . 83
5.2.2 Binary Instrumentation for Taint Tracking . . . 84
5.2.3 Data Flows: Access Control . . . 84
5.2.4 Taint Based Security Policy . . . 86
5.2.5 Example of Tag Propagation . . . 88
5.2.6 Network Marshalling . . . 89
5.2.7 Multi-Level Enforcement . . . 90
5.3 Timing Based Hardware Security . . . 90
5.3.1 Timed Key Usage at Hardware Module . . . 91
5.3.2 Requirements for the HSM and Discussion . . . 91
5.4 Related Work . . . 93
5.4.1 Exploit Prevention Techniques . . . 93
5.4.2 Information Flow Control . . . 94
5.4.3 Current On-Board Security . . . 94

6 Prototypes and Evaluations 97
6.1 Analysis of a CAN bus . . . 98
6.2 Protocol Measurements . . . 101
6.2.1 Simulating Key Exchanges with TTool . . . 101
6.2.2 Simulating the Secure Transport Protocol . . . 102
6.2.3 Implementation as Part of the Framework . . . 105
6.3 Distributed Dynamic Information Flow Tracking . . . 106
6.3.1 Performance . . . 107
6.4 In-Vehicle Prototype . . . 108
6.4.1 The CAN-Ethernet Gateway . . . 109

7 Conclusion and Outlook **111**
7.1 Achievements and Conclusion 111
7.2 Outlook on Future Research and Development 115

A Résumé Étendu — Français **119**

B Additional Measurements and Implementation Details **163**
B.1 HSM Performance Measurements 163
B.1.1 Performance Figures 163
B.1.2 Overhead of Prototype Implementation 163
B.2 Key Distribution Implementation 166
B.2.1 Application Code 166
B.2.2 Client Framework Code 167
B.2.3 Server Framework Code 168
B.2.4 Detailed MSC for Key Distribution 169
B.3 Secure Storage 171
B.3.1 Key Access Control 171
B.3.2 Native Encryption 171
B.3.3 Secondary Encryption 172
B.4 Intrusion Detection Sensors 173
B.5 CAN Ethernet Gateway 176
B.6 Active Brake Prototype Demonstrator 178

C Glossary **181**
C.1 Acronyms and Abbreviations 181

List of Figures **187**

List of Listings **189**

List of Tables **191**

List of Publications **193**

Bibliography **197**

Chapter 1

Introduction

> "The three golden rules to ensure computer security are: do not own a computer; do not power it on; and do not use it."
>
> Robert Morris Sr.

This Chapter provides an introduction to the application domain and goals of this thesis as well as its contributions towards solving today's problems in the field of embedded vehicular communication, in which vehicles have turned into *"computers on wheels"* [RH05].

1.1 Vehicle Electronics

In recent decades, vehicles were equipped with an increasing number of electronic controls. Without electronics, today's vehicles would no longer be able to comply with current emission standards and the driver's expectations of comfort and entertainment. They have become a vital part of the automotive architecture. This architecture consists of an internal network of small computers, so called Electronic Control Units (ECUs). Such ECUs may be part of entertainment or Human Machine Interface (HMI) systems to interact with the driver, or complement technical and mechanical systems. Every ECU is connected to a network as well as a number of sensors and actuators it is controlling. Typically, the sensor values are part of a closed loop that controls the actuators. We will give an overview on how vehicle electronics and on-board networks have developed in Chapter 2.

1.2 Motivation

The Connected Vehicle In addition to sensor and control loops that are an integral part of today's vehicles, the automobile has become an entity with various internal and external interfaces and connections. The vehicle connects to the internet for various online services as OnStar, ConnectedDrive, and many others. It no longer only plays digital music from physical media, but also from devices such as mobile phones using wired and wireless connections and various protocols. The recent trend of creating WiFi networks in vehicles for connecting mobile devices, brings yet another new communication stack into the car. This includes high-level protocols such as UPnP and DLNA. All these interfaces and protocols create an attack surface that can be used to mount attacks on the in-vehicle network. Even tire pressure sensors transmit their data wireless, thereby exposing another, albeit proprietary, interface.

Today, the primary threats targeting on-board networks are vehicle theft, odometer and firmware manipulations, component fraud, and unlocking paid functionality. We have however seen that devices with increased connectivity receive more attention from the malware industry, as for example mobile phones have become increasingly connected in the past and thereupon attracted attacks. A comparable development can be expected for the connected vehicle.

Road Safety In addition to the existing interfaces, a dedicated radio link at 5.9 GHz is currently being standardized for road-safety applications. These safety applications based on wireless communication are promising with respect to the reduction of fatal accidents. While communication-based safety scenarios introduce a new era of safety applications, new security threats need to be considered for successful application deployment. Besides safety scenarios, internet services and their seamless and intuitive integration into vehicles becomes an integral part of automotive scenarios.

These safety scenarios enable new application domains, in which new security threats are posed against the communication infrastructure between vehicles and infrastructure. The terms Car2X, or sometimes V2X have been coined for inter-vehicular communication. Solutions to secure communication between entities of Car2X scenarios, vehicles and Road Side Units (RSUs), have been investigated in recent years. However, security with regard to protecting against attacks on the on-board network has only been partially addressed even in research work.

Today, there exists a gap between what is possible to achieve with inter-vehicle communication and the trust required in the data received, originating from possibly vulnerable in-vehicle networks. This means that genuine pieces of software rely on information sent by other vehicles. This can directly influence the behavior of the vehicle, e.g., in an emergency brake situation. While information can be certified and cryptographically secured, a cryptographic signature and

certificate may only provide the assurance that the sender is authentic and the information has not been changed between the generation and the verification of the signature. However, no statement about the content can be made, i.e., *one is not able to assure the correctness of data*, as the sender or an earlier data source may have been compromised and could have generated false data before data was cryptographically signed.

While certificates with different assurance levels may provide trust in the sender up to a certain extent, the data source may still be compromised. Other security measures such as plausibility checks provide means to mitigate this risk of obvious attacks.

This problem is also found elsewhere: A cryptographic signature can only ensure authenticity and integrity of some data, but it does not assure the validity or the benign nature of data itself. An example are software bundles, where the distributor has to be trusted that a piece of software does what it is advertised as. Especially self-signed certificates do not provide authenticity at all, so that a signature provides only integrity protection for the download – and even this can be circumvented by man-in-the-middle attacks that change the certificate on-the-fly.

Loadable Code More and more platforms, mostly in the mobile world, allow the owner to install custom applications. In the early days of smartphones, for example, no security was integrated, e.g., into the palm platform or the windows mobile operating system. With an increasing number of threats like viruses and worms targeting an attack of embedded platforms, in particular highly connected mobile platforms, security now plays a major role for runtime loadable program code. We see a similar movement in the automotive world, where after-market customization of electronic features become available for clients. This ranges from weather applications at the head unit up to mobile applications on connected smartphones, accessing vehicle functions[1].

In contrast to common computer systems, automotive systems are highly safety critical and thus need adequate security protection, especially with the rise of a multitude of new attack surfaces. On the other hand, the vehicle platform is an embedded environment: Cost constraints, latency and function constraints are the primary concerns until now. Cost, for example is the main reason why asymmetric cryptography cannot be deployed on typical control units and sensors.

[1] In 2010, an iPhone application was released, allowing owners of the latest MINI vehicles to read out and configure vehicle data [BP10b].

1.3 Problem Description

In this emerging system of systems data is exchanged between domains: Boundaries become less prevalent and even user-added applications that access parts of the automobile are features in next generation vehicles.

Attack Contexts

We anticipate novel threats towards automotive systems coming into the vehicle via three new very exposed interfaces. Those *attack contexts* are broader than particular attack vectors. In addition to existing interfaces, such as OBD-II or the wireless interfaces mentioned in the beginning of this Chapter, the three novel attack contexts spawn a completely new surface for mounting attacks on the vehicle. Namely, these contexts are:

1. Vehicle to Vehicle or Infrastructure Communication Channels, used for safety and comfort functions.

2. User operated consumer devices accessing well-defined interfaces.

3. An execution platform inside the vehicle that can execute foreign code, e.g., user-added applications [Lut11, MTV09, GEN09] and on-demand ITS services [Kom10].

Figure 1.1 illustrates at which parts of the vehicle more or less openly accessible interfaces are located.

Especially the execution of code with unknown and untrusted origin poses a problem that can not be addressed by a single protection technique. In Section 2.2 we show different techniques and classify them with regard to complexity and security, and how some of them may be combined.

Scenario The three attack contexts are merely connected today. While applications within the vehicle already now have several distributed data sources and provide their data to other nodes in the vehicle network, they do not require on-line access. In the future, these data sources and sinks will be even more distributed, i.e., applications will access back-end services, ITS services, or internet services. Boundaries between these services diminish and after-market applications are integrated with vehicle internal services. Thus, the three attack contexts will grow together, posing potential threats to all of the domains.

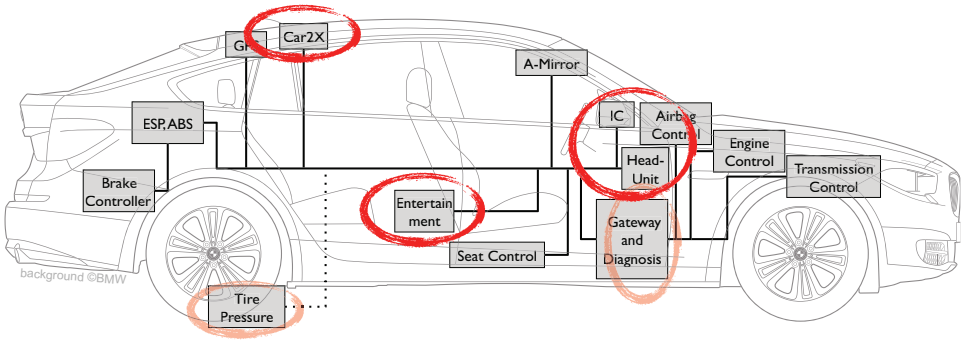


Figure 1.1: Novel Attack Contexts of the Connected Vehicle: Car2X, mobile devices, and loadable applications are the main exposed interfaces. Diagnosis and wireless interfaces (e.g., wireless tire pressure sensors) are currently not protected adequately. Vehicle in background from BMW.

Problem In the scenario, actors are heavily interconnected and may even change roles. Several domains, of which some are heavily restricted (the vehicle on-board network) and some are freely accessible (internet services), come together although they do not share common security measures. Currently, some security measures are only based on isolation (i.e., the non-accessibility of the vehicle network). Others are based on cryptographic measures and certification (authentication for internet servers) or security policies (firewall rules). The scenario thus cannot easily be assessed with classic security techniques, such as cryptographic signatures, static access control policies or trusted platforms. Due to cost and functional requirements, security techniques may not be applied to the overall vehicle platform and infrastructure. The electronic components of a vehicle naturally constitute a distributed system. Therefore, security must be regarded as a whole, and not isolated to a specific ECU. As information flows exist also across different domains, a security solution needs to be designed that complies to the security requirements of inter-domain application scenarios.

1.4 Goals

The overall goal is to mitigate the risk of attacks arising from the three attack contexts and their combination. A central unsolved problem of vehicle networks today is to maintain and attest the authenticity and correctness of the internal vehicle network between components, which is far more exposed than it was several years ago.

While the three attack contexts seem, at first sight, orthogonal, a combination of techniques may allow us to achieve the goal to prevent and mitigate attacks and ensure the system's privacy. The third attack context explicitly demands

for privacy and runtime security, as foreign code can be loaded at microcontrollers of the vehicle and access personal information, such as payment details or navigation history and vehicle services. Today we already see examples of data theft targeting mobile devices: Applications can secretly submit location data for advertisement companies, or take over the device itself in order to commit fraudulent payments or calls. This type of privacy infringement may as well affect vehicular platforms, which currently open themselves up for the same types of applications.

We have contributed our implementations and design approaches in the scope of the EU FP7 project EVITA. We describe the EVITA prototype environment that was used in the scope of the thesis work in Chapter 3. In particular, the software framework and the hardware security module were important building blocks for our implementations. As we specifically target the vehicle context, we emphasize some specific constraints and prerequisites for automotive communication and execution platforms.

Communication A classic approach to address the validity of received data is cryptography. Within the vehicle, there is currently no practicable technique to address this problem: ECU's microcontrollers do not offer enough processing power for additional cryptographic operations and communication buses do not allow for additional payload in data packets, a requirement for digital signatures or authentication codes. An additional constraint is posed by real-time requirements for certain automotive applications.

Platform While cryptographic mechanisms can assure the freshness, authenticity, and integrity of data, the receiver must have confidence that the sender's platform is in a trusted state and not compromised. This means that the runtime environment and the piece of software that generated the received data have not been modified by a third party. In personal computer systems, this is addressed by a technique called remote attestation, in which trusted hardware like a TPM chip, measures the integrity of running software. Although certain problems, for example regarding the privacy or the freshness of attestation may arise [CGL+11], we regard the approach as viable for attesting platform security within the vehicle. Due to technical restrictions and for reasons of cost (e.g., the need to use symmetric ciphers), remote attestation of in-vehicle components to secure communication between vehicles (Car2X communication) is out of question. For external Car2X communication, approaches either rely purely on cryptography in various forms (asymmetric [GFL+07], group [GBW07], timed [HL06]), or on dynamic reputation information [Ray09] as additional indicator.

Contributions

We have developed techniques and protocols for vehicular on-board networks, which can assure the authenticity and freshness of data (security), as well as the actual use of data (security and privacy) within the vehicle's network and inside control units.

1. Specification and Design
    - An on-board key distribution protocol.
    - A distributed IDS for vehicle networks and execution platforms based on Dynamic Information Flow Tracking (DIFT).
    - A policy-based approach for DIFT to describe allowed information flows.
2. Simulation and Evaluation
    - Evaluation of latency and error rates of the transport protocol on simulated CAN nodes.
    - Overall performance of key distribution on simulated CAN bus and on actual prototype Ethernet implementation.
    - Prototype evaluation of the distributed information flow tracking approach.
3. Prototype Implementation
    - Key distribution, integrated with the HSM and the framework.
    - Secure communication for CAN and Ethernet, integrated with HSM and the framework.
    - A CAN–Ethernet gateway.
    - An Intrusion Detection System for the vehicle, integrated with the framework.
    - Distributed Dynamic Information Flow Tracking (DIFT) based on local taint engines using binary instrumentation and taint propagation via secure communication links.

1.5 Structure of the Thesis

This thesis is divided into seven Chapters. Following this introduction, Chapter 2, the *State of the Art*, gives a comprehensive summary of current industrial practices and research approaches for both, the vehicular domain and other embedded computing environments such as mobile phones or computers. In Chapter 3, *Environment*, we introduce scenarios, which we have chosen to exemplify the approaches developed in this thesis. We provide an attacker model and a description of the vehicular experimentation environment, the European FP7 project EVITA, in which most of the work of this thesis was done, and to which it contributed.

This thesis continues with the two major Chapters: In *Communication Security*, Chapter 4, a protocol for Key Distribution in automotive environments and a protocol to establish secure communication on the CAN bus, are presented. Subsequently, Chapter 5, *Dynamic Platform Security*, assesses the overall dynamic security of the automotive platform: An approach that is based on dynamic information flow tracking to leverage runtime security. It also allows to restrain privacy infringing software. In addition, a classic intrusion detection system that is integrated with the vehicular security framework is presented. In addition, another approach that is suited to secure low-performance ECUs is introduced.

In Chapter 6, *Prototypes and their Evaluation*, we present simulations and measurements of our prototype implementations, as well as selected design decisions and details of the implementation.

The thesis concludes with the *Conclusion and Outlook* in Chapter 7. We discuss the feasibility of our approaches to secure the automotive on-board network, the applicability of individual techniques and their benefits and drawbacks. We also give an outlook on future work to the topics covered in the thesis, as well as perspectives in research and industrial development.

In Annex B, we provide *Additional Measurements and Implementation Details*. In Annex C.1 (page 181) *Abbreviations and Acronyms*, which are used throughout the thesis are expanded and explained. In the electronic form of the document, you can easily navigate to this page by clicking on an abbreviation.

At the end of the thesis, you can find lists of *Figures, Listings, and Tables*. A list of publications during the work on this thesis is given thereafter. The *Bibliography* which lists references to cited work completes this thesis.

Chapter 2

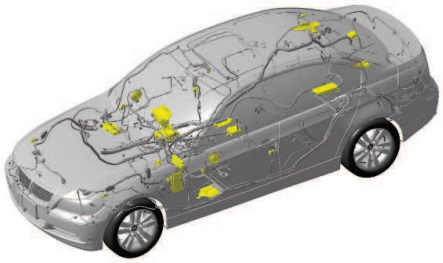
State of the Art

> "If I had asked my customers what they wanted, they'd have said a faster horse."
>
> Henry Ford

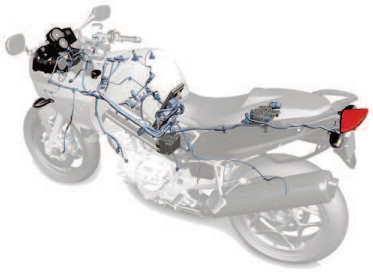
We have divided this Chapter into different parts: In the first part, we discuss the current state of the art in the vehicle domain. We give a broad overview about the domain of in-vehicle communication, in particular how today's "computer on wheels" has evolved from the mechanical automobile. Recent related work in relation to enhancing information security in this domain is explained and discussed in the first Section.

We then present a state of the art in software and embedded systems security with a focus on classic intrusion detection and containment approaches. Only a few solutions of these approaches can directly be employed in the automotive world. In the embedded system 'automobile', a different communication paradigm is used, latency requirements are strict and microcontrollers and networks cannot cope with additional overhead. We classify existing solutions with regard to complexity, overhead and applicability to the automotive domain, and show for which environments the different techniques are currently used. We deliberately take a broad view of security issues in related fields like security in mobile applications (e.g., smartphones), since a number of solutions, which may be appropriate also in the automotive domain, have been proposed or implemented in other domains. For the same reason we also discuss recent work in car-to-car security, albeit the focus of this thesis being in-car security. Indeed the vehicle domain is not fundamentally different with regard to onboard infotainment systems: Android is the operation system of Renault's R-Link: An App-Store enabled head unit introduced in December 2011[1].

[1] Market introduction is scheduled for November 2012.



(a) Relevant sensors and control units for $eCall$, an EU-wide emergency system [Eur09a], shown for internal networks of BMW's L6 architecture.



(b) CAN bus and electronic components of BMW F800 motorbike.

Figure 2.1: Today's networks consist of multiple control units, subnetworks and interface with external devices and networks. Shown is only a fraction of all control units for the vehicle, in particular those which provide relevant data for an eCall emergency notification. Pictures from BMW press portal [BP10a].

The strong motivation for security in vehicular environments arises from increasing complexity of electronic infrastructures and availability of communication interfaces for users aboard vehicles. This means in particular that consumer devices can be coupled with on-board networks [BP10b]. Standard hardware, such as Ethernet and IP, are integrated in today's vehicles [Jon10, JH08] next to USB and Bluetooth. In research, even the feasibility using IP as the main network protocol is examined [GHM+10]. Due to the fact that these interfaces are standardized and available to everyone, including hackers with criminal intent, attacks on vehicles are becoming increasingly likely. In Figure 2.1 a part of today's on-board networks in an automobile and a motorcycle can be seen.

With increased financial incentive, the risk of successful attacks increases likewise. Already today, odometer manipulations or pirated navigation material are a growing market for criminals. Manipulations of the on-board network to overcome legal restrictions also exist. An example of this is the injection of fraudulent velocity signals in order to watch TV or DVD while driving.

2.1 Automotive Security

In this Section, we give a comprehensive overview of the state of the art of software and network security in vehicular systems today, and show ongoing and future developments.

2.1.1 Vehicle Components and Vulnerabilities

Vehicles have traditionally been a mechanical domain. In recent decades, this has changed drastically. Starting with electronic engine management in the 70s, vehicles have evolved to a multi-connected computerized platform. At the same time, safety systems that are not only based on mechanical but also on electronic equipment (electronic stability, anti-lock brakes) have been introduced with great success. Vehicle-to-vehicle systems take a first step towards autonomous driving. Although a commercial deployment of completely self-driving vehicles is unrealistic, recent DARPA challenges for autonomous driving have impressively shown the feasibility [MBB+08, TMD+06] and actual on-the-road tests have been conducted in a larger scale between 2010 and 2012 [Mar10]. This led to an official license to be issued for the road use of autonomous vehicles in the state Nevada in May 2012 [Slo12] and for California in September 2012.

Electronics Sensors and control units have become increasingly complex in the last years. This is also due to the fact that redundancy plays an important role for risk-mitigation in safety considerations. An example for placing redundant sensor is the 'brake-by-wire' setup: there is not one main physical sensor at the pedal anymore, but a set of multiple redundant sensors, which measure different physical values. One of the sensors is used for the measurement of the brake pedal *tilt* and another one for sensing the pressure applied to the pedal. Such sensors are usually integrated into a sensor box connected to the vehicle's buses. An example implementation of an electronic brake pedal sensor box is described in [ISS02] from 2002. We have schematically depicted it in Figure 2.2. While the gas pedal is electronic in most of the vehicles sold today, electronic brake systems without mechanic fallbacks do not yet meet legal requirements in all markets. A flaw in the brake unit's software was the cause of a major recall of Toyota vehicles in 2010 [Wil10]. In October 2012, Nissan announced to introduce a steer-by-wire system in serial production for Infiniti vehicles within one year [Nis12]. Their system uses a setup of three redundant ECUs and a mechanical backup system to electronically control the vehicle's steering system, and give feedback to the steering wheel only selectively. The mechanical steering column can override electronic signals by using a clutch, which decouples the steering wheel from driving wheels at normal operation. Since 2005 such a backup would not even be required according to a revision of the ECE regulation that particularly embraces steer-by-wire systems [Uni05].

Current research topics, which in particular focus on enabling e-safety services are covered in this Section. In the near future, safety services and applications will be deployed in vehicles as well as in roadside infrastructure. This is envisaged by the Car-to-Car Communication Consortium or the European Telecommunications Standards Institute (ETSI) standardization body.

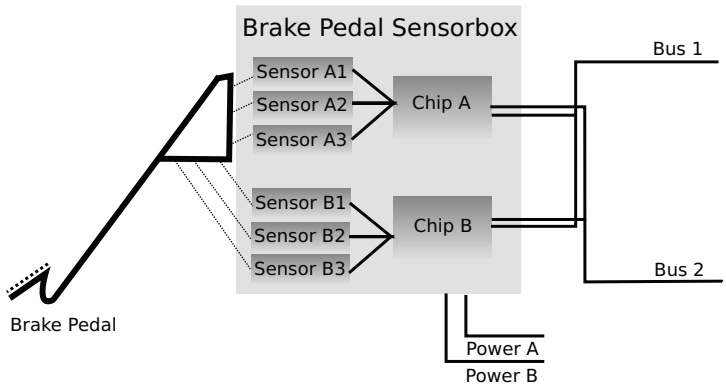


Figure 2.2: Multiple physical sensors (A1–A3 and B1–B3) are attached to one brake pedal for redundancy. The microcontroller chips, network transceivers and power supplies are redundant, cf. [ISS02].

In order to provide a trusted base for inter-vehicle communication, secure onboard communication for vehicles is crucial.

In first generations of multi-connected communication, vehicles will use information provided by RSUs and will be aware of other vehicles in the vicinity, thus being able to warn the driver of upcoming dangers early and adequately. While typically proprietary security solutions have typically been applied for classic vehicle functions, such as digital odometers and physical access control (e.g., key fobs to open the doors using wireless radio communication), inter-vehicle communication must rely on well-defined standards that have to be acknowledged and adhered to by all manufacturers and suppliers.

These novel applications provide new surfaces for attacks, which may even be exploited remotely and can result in serious consequences. Without additional security measures, false alerts can be created and attackers may exploit safety applications for their benefit, thus endangering the safety of drivers. We show examples of successful attacks to illustrate the need for security.

Attacks The current in-vehicle network architecture, which is used to exchange data between sensors and control units (ECUs), has grown over time. The industry has not yet seen the need to secure buses as they were not connected to the user domain or to the outside of the vehicle. However, experiments on vehicle bus systems show the severe impact an attack may have [KCR+10]. Similar experiments have been conducted earlier by [HKD08b]. The roadside infrastructure that exists today for traffic management and tolling systems has also successfully been attacked: In 2007 it was shown how TMC [TIS] over RDS may be manipulated with limited effort [BB07]; in 2008 weaknesses of the California FasTrak tolling system were demonstrated [Law08]. Both attacks were based on reverse engineering the systems and revealed that no cryptographic

security was present.

While low-profile privacy infringing attacks have been successfully mounted on cheap radio sensors, such as tire pressure sensors [RMM+10], more sophisticated attacks have succeeded to extract cryptographic material of remote controls with the help of timing and differential power analysis [IKD+08, EKM+08]. Even where cryptography is in place, relatively simple attacks such as relaying radio signals over analog channels (due to latency requirements) can lead to successful attacks and break-ins on vehicle anti-theft components. How to unlock and start a vehicle has been successfully demonstrated on a large number of vehicles from different manufacturers in [FDC11].

In general, we can distinguish between two types of attacks:

1. Attacks on the integrity of the vehicle.
2. Attacks to infringe the driver's privacy by means of collecting and tracing personal data at vehicle operation.

We do not take attacks aimed at stealing a vehicle explicitly into account but regard it as a subset of attacks on the vehicle's integrity.

Vehicle Interfaces As shown in the previous paragraph, vehicles offer a variety of interfaces already today. With the introduction of WiFi and Ethernet in current vehicles [Jon10, Bru10], vehicle interfaces are gaining more attack potential, as no special knowledge is required to access such interfaces. We believe that especially the integration of consumer-oriented interfaces needs to be secured. The following list shows which services and interfaces are exposed, and thus prone to be attacked.

- Mobile Devices (Device Integration of Consumer Devices for Infotainment)
- Diagnosis and Reprogramming (Workshop Access and Maintenance)
- Remote Services (OEM back-end accessible through network connectivity of vehicle)
- Applications/Widget (Small programs being run on vehicle computers, not yet on the market)
- Car2X (Safety and Service applications, not yet on the market).

All of these domains have in common that they explicitly involve a great level of interaction with the vehicle's on-board network. While diagnosis and flashing (reprogramming) require a deep access to vehicle functionality, infotainment devices only need limited access to enable commodity functions, e.g., controlling

the heating or A/C remotely. While a separation of such functionalities seems trivial at first, current vehicle network architectures ignore this fact [KCR+10].

Recent trends in automotive development move towards the centralization of electronic control units. This means that instead of a multitude of single-purpose ECUs, fewer multi-purpose ECUs will handle a variety of tasks. A side-effect of this centralization is that different functionality on such ECUs need to be isolated or separated, e.g., by means of virtualization approaches. While there exist approaches for other embedded devices, such as cell phones, this approach is relatively new to potentially safety critical systems. The trend will most likely initially be adopted for infotainment systems, as envisaged in [GEN09, HH08].

2.1.2 Vehicle Security Concepts

VANET Security After the first cooperative vehicle-to-vehicle communication systems were successfully tested in the early 2000s [FFH+04], the need for security has become evident, as successful attacks on safety functions may not only perturbate the system, but can lead to fatal accidents. The SeVeCOM project [SeV08] assessed this need for security and conducted several experiments in 2006. It turned out that certificates, which are needed for communication partners to authenticate remote vehicles, cause the main overhead in communication. Thus Elliptic Curve Cryptography (ECC) was chosen as preferred algorithm. It features a shorter key and signature size, but offers the same security level as RSA using longer keys [LBH+06]. Optimizations of channel usage, e.g., by intelligently omitting certificates if possible [Pap09], have been developed in this project. The network of excellence in IT system security, SysSec, has published a report on the state of the art of security in the connected vehicle [Sys12]. It covers Car2X as well as in-vehicle security.

In-Vehicle Security Although it is mentioned in many vehicle security projects that in-vehicle security systems must eventually establish the necessary trust for cooperative applications [Kun08, EB06, GFL+07], none of these projects have investigated possible solutions and the implications of a secure in-vehicle platform. The EU project EVITA aims at exactly this: provide security and trust inside the vehicle. Already at sensor-level it is necessary to add security measures, because in-car data will possibly activate vehicle-to-vehicle applications. Thus internal messages, such as "airbag deployed" or "emergency brake", can cause warning messages to be broadcasted outside the vehicle.

Security measures have so far been applied for very specific functionality and in a few vehicle components. For example the upload of firmware updates is protected by an access code of two or more bytes [ISO00]. Of course, a two-byte code only provides marginal security against attackers with unlimited time and trials. In [KCR+10] it was shown that the security code required by

the standard can be broken in as little as three and a half days by performing a brute force trial-and-error search. The introduction of strong cryptographic signatures has recently been agreed between a group of manufacturers (HIS) in Germany [ZS08].

2.1.3 Ongoing Research

Field Trials Field Operational Tests (FOTs) are programs for evaluating the feasibility and performance of inter-vehicular communication in real-world test beds. In FOTs, a number of vehicles, usually between 10 and 200, are equipped with on-board systems and are operated in a specific area that is equipped with RSUs. In all current FOTs, cryptographic security for wireless communication is applied. For example the project sim^{TD} uses a public key infrastructure for long-term identity as well as short-term identities, which provide pseudonymity. These are established and operated [BSM+09] through a Public Key Infrastructure (PKI) that was established by the project consortium. The concept of short term identities is related to the direct anonymous attestation approach, as found in Trusted Platform Module (TPM) architectures. Due to limited processing power, the FOTs usually fall short in the cryptographic part of protocols. For example, instead of elliptic curve cryptography, which is anticipated for standardization, RSA-512 is used for short term certificates in sim^{TD}.

Similar FOTs exist in other countries, e.g., SCORE@F in France. These FOTs are coordinated through the Drive C2X project, which conducts interoperability tests towards standardization in ETSI.

In-Car Security In the scope of EVITA, it was one goal to associate in-vehicle security systems together with external communication and corresponding safety applications, as well as provide a base for future projects concerned with in-vehicle security. The EU project OVERSEE focuses on secure vehicle runtime environments [GHR+09] and the German project SEIS is going to securely implement the Internet Protocol (IP) into automotive on-board network architectures [GHM+10]. Recently, some new approaches to protect CAN bus security have been proposed [CRH05, NLJ08, OYN+08, GR09, Be09, HSV11]. The FlexRay bus, which is believed to succeed CAN, shows very similar shortcomings in security design, as specifically eavesdropping and injecting payload are still possible [NLPJ09]. Some approaches require adapted transceivers chips, and some even different network media to be deployed. All of the approaches use cryptographic security to authenticate the bus payload. We will explain these approaches in more detail in Section 4.3. Overall, the security of in-vehicle components has gained interest among researches and industry. This is also illustrated by the fact that security consulting and anti-virus companies publish survey and white papers on this topic [Lin09, MWW+11]. A good and comprehensive overview about current research activities related to in-vehicle security

is given in the survey paper [KOJ11]. The SysSec report presents details on the state of the art of security in the connected vehicle [Sys12].

Car2Car Security The outcome of FOTs will largely influence standardization and industrial feasibility. The European Commission has issued a mandate to the standardization bodies ETSI and CEN [Eur09b], which covers the standardization of vehicular communication and its applications. In the United States, VII and VSC (Vehicle Infrastructure Integration and Vehicle Safety Communications, funded by the US Department of Transportation, USDOT) projects have already resulted in the IEEE WAVE standards [HL10, US 09]. The current state of standardization and challenges in the US is presented in [LLZ+08]. A modified form of the WAVE (IEEE 1602.11) specifications are used for FOTs in Europe. The European Commission has given a mandate for developing Car2X standards to the standardization bodies CEN and ETSI [Eur09b]. They are therefore likely to be acknowledged by standardization. Apart from vehicle-to-vehicle systems, the emergency eCall system [Eur09a] is going to be mandatory for new vehicles within the European Union. We are going to see more and more integration of communication hardware and software into vehicles in the coming years, which demands for increased security awareness.

2.1.4 Current Practice

Car2X Communication Despite the fact that Car2X communication solutions are not yet deployed in today's vehicles, there exist best practices that have been the result of many research projects and field tests [FFH+04, PBH+08, SKL+06]. While the focus of this thesis is on-board communication, this also plays a role in the trust-establishment for Car2X systems. Currently, a certificate-based approach with different trust levels is envisaged.

In Car2X systems, there are mainly two types of messages called CAM and DENM. The beacon messages are called Cooperative Awareness Messages (CAMs). These beacons are periodically broadcasted and announce the vehicle's position, speed, and acceleration. The CAM beacon messages are used to avoid collisions: A trajectory forecast is calculated for the surrounding vehicles, and compared to the own vehicle's trajectory, so that the driver can be warned if a potential collision scenario arises.

The other type of messages are so-called Distributed Environment Notification Messages (DENMs). They announce localized information such as danger warnings that are generated by a sudden brake maneuver or if a crash has already been detected. These messages are also used to broadcast road and infrastructure condition and status information from road side units such as traffic lights. While CAM messages are always consumed at the receiver, DENM messages

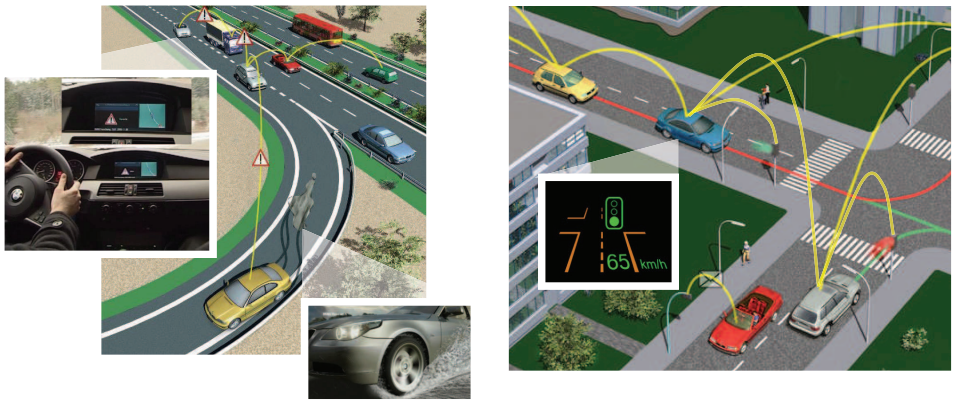


Figure 2.3: Car2X Communication examples: DENM messages are emitted and relayed to warn other drivers (left) or to enhance traffic efficiency, for example by broadcasting traffic light schedules (right). Pictures used with permission from BMW Forschung und Technik.

can be relayed to other vehicles, e.g., for conveying a warning message upstream along with cars in opposing traffic.

An important factor in wireless communication is securing the content appropriately: As mentioned above, ECC is envisioned for the cryptographic signatures in Car2X communication, due to shorter key length and signatures. A certain feature in Car2X is that pseudonym certificates are used for signing data that is sent: The identity of the vehicle can not be easily revealed. This preserves the privacy of drivers and has been assessed in the EU project PRESERVE [PRE14]. Each vehicle holds a long-term identity certificate. This identity is used to authenticate against a back-end PKI in order to retrieve fresh pseudonym certificates, which are then used on the road.

The current state of research in vehicular networks is described in [HL10]. The book covers all communication layers and includes a Chapter about Car2X security approaches and emerging standards.

Communication Modeling The current development process for distributed application over multiple ECUs is historically reasoned. When the CAN bus was introduced, it replaced dedicated electric wires with voltage or otherwise coded signals by a computer bus, where digitalized information is transmitted. Thus, the data on the bus is often referred to as *signals*. These signals are typically transmitted frequently in a periodic manner (typical cycle times are 10 to 100 ms), which allow closed loop controls.

During development, these signals are defined in software tools that are available from different vendors (Vector's suite being one of the commonly used, and RTaW-Sim being a free alternative). The developer adds signals and their

corresponding discretization representation. These signals are then bound to sender ECUs and one or more receiving ECUs are selected. The resulting matrix is referred to as the $K\text{-}Matrix$. It described the representation of data, and the sender/receiver-group relation. ECUs with multiple network interfaces can be enhanced with forwarding rules and hence become $gateways$ within the network.

The modeled data can be exported in specific formats describing the bus communication. Am XML-based format called FIBEX (Field Bus Exchange Format) is currently becoming an industry standard. It was defined by the ASAM standardization body (Association for Standardization of Automation and Measuring Systems).

Until now, the tools to model communication do not include means to enhance the payload with authenticated or encrypted data. While the information about communication groups, message frequency and size are explicitly available, they are also not yet used for generating or defining policies for enforcing access control on the network or detecting malicious and non-conformant data.

AUTOSAR The AUTOSAR architecture is based on a paradigm, which has been promoted for use in vehicle networks since the mid-2000s. It comprises an embedded operating system (AUTOSAR OS, based on OSEK, an ISO-standardized automotive OS [ISO05]) and an abstract view on all communication, i.e., the concept of an abstract Virtual Function Bus (VFB), which replaces explicit inter process communication and network data exchange. This model-driven approach allows to deploy software components (SWC) on different distributed components, the ECUs at an abstract level for a "push button" compilation and deployment. The operating system provides a framework for software functions and tasks that can access hardware and libraries through a standardized programming interface. Figure 2.4 shows the development paradigm, which compiled all models into one deployment architecture.

Since the last major release, AUTOSAR 4.0, a cryptographic service library is part of the OS specification [BFWS10]. This library provides functions to applications and OS functions for specific algorithms: MD5, SHA-1, RSA, AES and others can be accessed via the AUTOSAR CryptoAbstractionLibrary [AUT11].

Despite the standardization of the Crypto API, there is not yet a way to secure communication between AUTOSAR communication links. In particular, if secure communication is to be implemented, the security payload, e.g., an authentication code, would need to be added to the application layer. This is currently done for system functions such as firmware flashing or certain diagnosis functions.

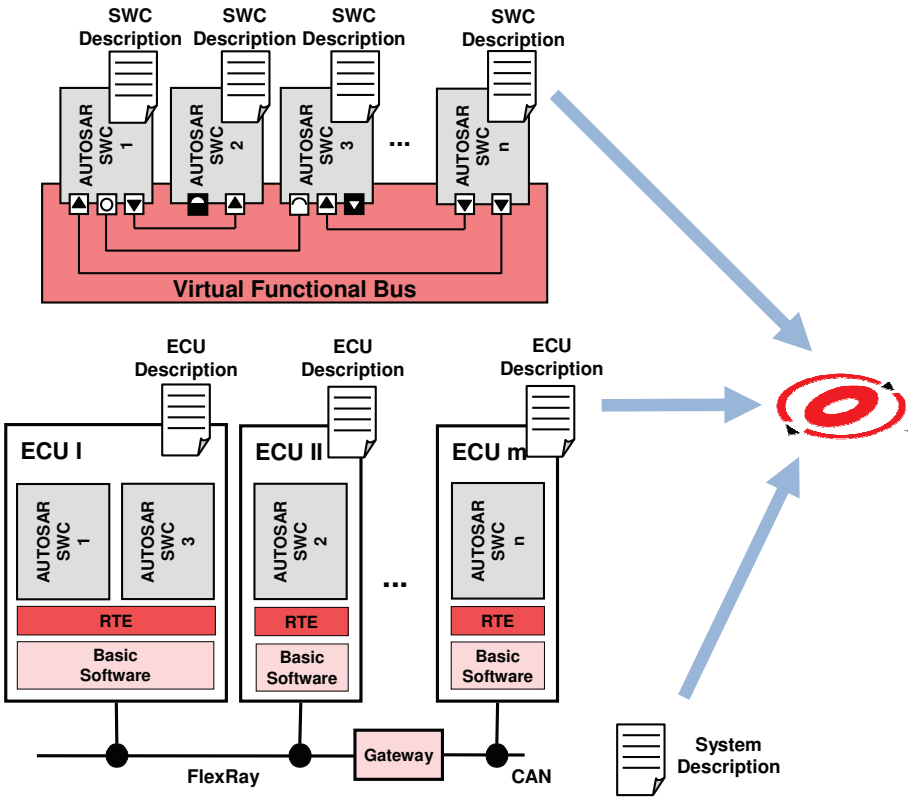


Figure 2.4: AUTOSAR Development Paradigm: Software Components (SWC) exchange information via the Virtual Function Bus (VFB). These software functions are deployed on different ECUs. Models are described in a machine readable format and compiled into the overall vehicle architecture for deployment. The framework's code is partly generated from the so-called "AUTOSAR configuration", which comprises all description files. Illustration from [BFWS10].

2.2 Software Security

This second part of the current Chapter provides an overview on related work with respect to security mechanisms for preventing and detecting attacks and intrusions, as well as approaches to intrusion response and containment. In order to achieve these goals, IT security research has been developing solutions relying on various approaches serving different security requirements and constraints of the dedicated application environment. The need for security in classic computer systems has evolved from open networks and systems with closed user groups (e.g., research-only ARPAnet) towards an open user group (e.g., today's internet, where code is executed within the user's web browser) and closed systems with applied authentication and other security solutions.

In the scope of application security, there are several known approaches for the prevention, detection and containment of attacks:

- Software Trust Management, i.e., certification of software based on:
  - Security-optimized programming techniques
  - Formal verification of software of source code
  - Proof-Carrying Code
  - Software testing
  - Digital signatures and a Public-Key Infrastructure ("code signing")
  - Reputation information
- Access Control Mechanisms:
  - Sandboxing
  - Policies for I/O and File Access
  - Packet Filters
- Intrusion management including intrusion detection and response. These techniques provide measures based on:
  - Stateful inspection of data analyzing the plausibility, or matching to pre-defined heuristics
  - Triggering of pre-defined actions in order to contain an attack or even disarm the attack
  - Communication and information flow, based on defined rules or learned characteristics.
- Static Information Flow Analysis of:
  - Source Code

  - Binary Data

- Dynamic Information Flow Analysis of:
  - Measured data, user input or network communication
  - Execution of programs, using hardware or software support, such as symbolic execution or on-the-fly binary instrumentation, or altered computing architectures.

- Hardware Security Support:
  - Secure Boot / Platform
  - Secure storage of key material and execution of algorithms and

The majority of the solutions mentioned above is not applied on its own. If applied correctly, security solutions offer a higher level of protection when they are combined. Some of the solutions mentioned above are complemental. After giving an overview of the available techniques, we show how they are usually combined and how well they are suited for different scenarios.

2.2.1 Overview and Classification of approaches

Out of the multitude of different approaches to detect and address attacks and implicit information disclosure, we present a selection of techniques with regard to the program code's life cycle. In Figure 2.6, we show the different security measures and their applicability.

The techniques range from static analyses of the source code to dynamic run time analyses on the target execution platform. While detailed and in-depth analyses can be performed using the source code, in practice the source code often is not available. On a given target platform, code of unknown origin shall be executed in a timely manner. Thus, a sophisticated analysis of binary code is not feasible. This is especially true for program code that is executed in a browser, on a mobile phone or on the head unit of a vehicle. Security techniques aim at establishing trust of the owner towards a piece of program code obtained from selected or arbitrary sources.

A common technique to facilitate checks of code integrity are certificates, which are issued by trusted instances of a PKI. It should, however, not be seen as a single security measure, as it can only guarantee the non-modification of program code and neither the flawlessness, nor the correct behavior of program code.

> Certificates aren't like some magic security elixir, where you can just add a drop to your system and it will become secure.
> C. Ellison and B. Schneier in *Ten Risks of PKI: What You're not Being Told about Public Key Infrastructure*, [ES00].

This quotation is not only true for certificates but for nearly all security measures and techniques. System security is only as strong as the weakest point of enforcement. Thus, it is sensible to combine different techniques at different points in the lifetime of program code.

In the following paragraphs, we show the strengths and weaknesses of different techniques. Many of them are widely applied, while some have not found application due to heavy constraints.

Code Signing The widely used approach to overcome the problem of executing code of unknown origin is a trusted third party, the certificate authority, which authenticates program code in a process called *code signing* [Fle02]. The trust towards this party is usually established using a PKI. This approach, however, does only guarantee the authenticity of origin of a certain piece of software. If the origin is unknown or untrusted, the code may very well be malicious despite the fact that it is correctly signed. Only if the certification step involves an analysis of the software, this technique may offer a certain amount of security towards unknown origins.

Restricted Execution Environments: Sandboxing The execution environment itself can take a number of measures with regard to unknown attacks and unknown code. A standard method, which has been used since 1993 (4.2BSD) on Unix operating systems (under the names of *chroot* and *jail*) and internet browsers and Java interpreters (under the name of sandbox [WLAG93]). This approach restricts the abilities of software by means of system calls, I/O (file system and peripherals) as well as memory access. It is also referred to as "software-based fault isolation" [WLAG93]. This principle has been taken up for code of unknown origin by runtime environments with intermediate object code. Java and .NET use runtime restrictions for programs loaded in an untrusted context such as web-applets.

Proof Carrying Code Proof Carrying Code (PCC) is an approach, in which source code is programmatically proven to conform to certain properties. These properties can be memory related (e.g., only reads and writes from and to certain areas) and run-time related (e.g., that the code only uses certain processing

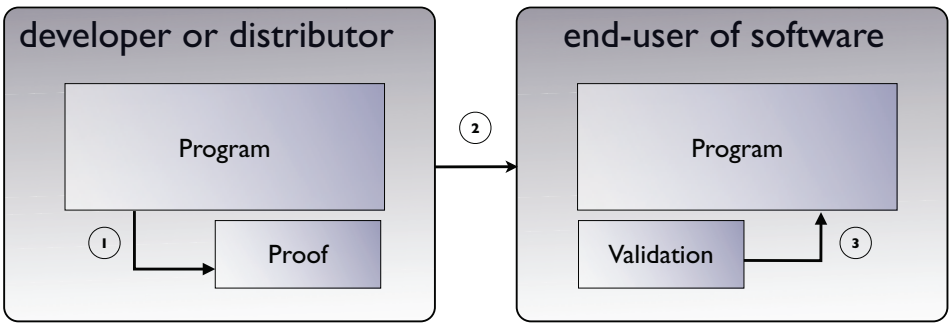


Figure 2.5: Partitioning for applying Proof Carrying Code: A set of proofs towards the program is generated at programming or distribution time (1). This is typically the most complex and expensive step. The code is distributed to the user (2). Before execution of the code, the proofs are evaluated on the actual code (3). No cryptography needs to be used, as changing the proof set or the program code results in failed validation.

time). It has been established by George Necula and is detailed in [Nec97] and [NL98] as well as his thesis [Nec98].

Using PCC, the developer of program code provides a formal proof towards the fulfillment of certain security properties. This proof is machine-readable and can be evaluated, i.e., reconstructed, in short time. This is done on the target platform by a so-called "theorem prover". Figure 2.5 shows the partitioning between producer and consumer.

This technique uses no cryptography at all. Thus, it does not rely on a trusted third party or shared secret and is self-contained. Changes of the code or the proof are immediately detected by the theorem prover before execution.

Code that is to be executed always has to be proven, before it can be distributed and loaded onto a target platform. For certain properties, such as type safeness, such a proof can be constructed automatically [Nec98]. However, for more complicated properties, the proof itself may be complex. Also, it can not be foreseen which particular proofs the target system may be interested in. Many properties are not easy to prove or not provable at all, such as the halting problem.

Model Carrying Code In [SRRS01] and [SVB+03] a framework is described, in which code providers attach a model of the code's behavior. This technique is called Model Carrying Code (MCC). The model is described in a language called BMSL, which covers security related interactions of the code. These interactions are matched with a policy set of allowed actions at the target side. Additionally, these policies are enforced and monitored in a restricted runtime environment.

The behavioral model described in [SVB+03] is derived from *"the security-relevant operations made by the code, the arguments of these operations, and the sequencing relationships among them"*. For a concrete implementation, this means that system calls and library calls are extracted from program code and compiled into a behavioral model. Approaches to classify short sequences of system calls have been successfully tested [HFS98]. Comprehensive behavioral models that are matched and verified at execution time offer an additional step of security against modification of code, without the requirements of additional infrastructure as would be required for PKIs. The MCC approach explicitly mentions cryptographic techniques and proof-carrying code as sensible enhancements to strengthen security.

Abstraction-Carrying Code Abstraction Carrying Code builds up on the concept of Proof Carrying Code, but does not supply concrete proofs but abstract properties that are certified by the distributor of code. These abstractions may be validated at the target system. The approach presented in [APH08] has been tested in a prototype implementation for the Java framework [XH03].

Code Attached Requirements Recently, a different yet somewhat related technique popped up alongside the Android security model. For this platform, not only classic mechanisms as code-signing and restricted execution environments are used: Programs need to supply a set required permissions. These are attached to the code and matched with user-given policies, i.e., access must be explicitly granted if not allowed by system-wide policies. In contrast to PCC and MCC, the properties do not relate to abstract properties such as type-safety, but to specific access control rules (e.g., access to location, address book and internet functionality). On the one hand this approach is very useful, as it allows the user to select which resources may be used (e.g., a weather application would be suspicious if it accesses the microphone, as shown in [Hig10]). However, on the other hand, the implicit usage of data can not be controlled by the user anymore. This means that location information, which should, for example, only be used to determine closest weather station of a user, can also be given to any other arbitrary party on the internet, including advertising companies. A recent study analyzed implicit information streams for Android mobile applications, finding that 20 out of 30 applications silently disclosed private data to third parties [EGC+10].

Stack Layout A problem on lightweight microcontrollers, often used in embedded systems, are stack overflows. These may not only be caused by buffer overflows (failures in software) but also by the lack of memory—a scarce resource on these systems—when the stack grows too big. Additionally, such stack overflows may go unnoticed due to the absence of a memory control unit.

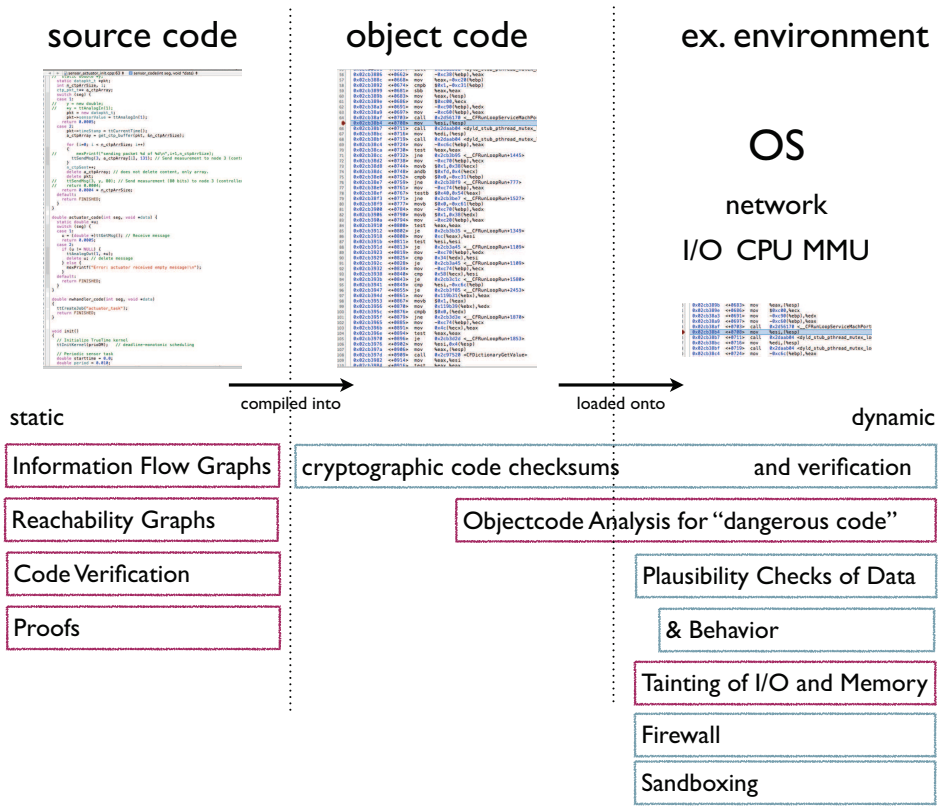


Figure 2.6: Program code and different security measures that can be taken during programming, compilation and execution.

In [FPC09] a solution of this problem is presented by Francillon et al.: the data stack and return addresses are separately stored in memory, which makes overwriting the return pointer impossible. Operations that directly or indirectly change the CPU's program counter are restricted. They have validated their approach based on an AVR Atmel core in software simulation and in an VHDL hardware design and show that the approach comes at little extra cost and complexity.

Plausibility Checks A technique, which is widely used for safety and malfunction detection, is plausibility checking: Data are examined for validity by means of domain-codomain characteristics (e.g. $\forall t \in \text{Temp}_{ext} : -50 < t < 60$ with Temp_{ext} being external temperature sensors, measured in centigrade). Often, multiple independent data sources are used to determine plausibility using a voting-based approach. This technique can be adopted for runtime security checks, which need not to be limited to data plausibility based on thresholds, but may take temporal and behavioral patterns, such as memory usage or pro-

cessor usage patterns into account. Plausibility-based anomaly detection is an integral part of behavior based intrusion detection systems.

Reputation Information One possible technique to address reputation is a consensus-based mechanism, in which individual agents $rate$ an object with regard to different properties (usability, adherence to requirements, trust). This object can be an entity, a service, or a piece of software. This mechanism demands support from the infrastructure and imposes latency and communication overhead. While this approach is viable for online-services and dependability problems, such an approach has also been proposed for generic loosely coupled systems [Wey08] and in the vehicle context [Ray09].

2.2.2 Techniques and Methods for Information Flow Tracking

Several domains make use of information flow tracking. While this technique normally generates a large amount of overhead, it permits a fine-grained analysis of data and control flow throughout the execution of inquired code. Fields of application include (cf. [GGZ+08]):

- Debugging (bug tracking and fault avoidance)
- Security (software attack detection)
- Data validation (lineage tracing of scientific data)

In the following, we focus on the use of information flow tracking for security purposes, i.e., to ensure privacy of data (lineage tracing) and to detect attacks by means of intrusion detection systems. We give an overview of current approaches in the academic and industrial domain.

Code Shepherding Code shepherding is a technique described in [KBA02], in which multiple measures are taken against malicious code. The origin of code results in a certain trust (or distrust) in code. Data are marked (tainted) and followed throughout the execution. Additionally, sandbox like behavior is enforced and monitored: Only previously known entry- and exit-points for shared libraries are allowed. A technique to dynamically introduce check-points into running code is used to validate properties and strictly enforce sandboxing. The tool they used is called "RIO", which operates on IA32 binaries in PE and ELF and can thus be used on Linux and Windows platforms.

Malware Analysis utilizing Information Flow Techniques In order to analyze unknown binary code fragments, as they appear in worms and viruses, information flow techniques have become an interesting and helpful tool. There exist static approaches that analyze the binary as-is, and dynamic approaches that examine the runtime behavior of such malware. This is often beneficial, as binaries are often obfuscated by encoding, packing, and self-modifification. In [BMKK06] an automated malware analysis system based on emulated execution was developed and tested. Recent developments in in this field are discussed in [ESKK08].

Dynamic Information Flow Tracking DIFT is a technique, by which input data (e.g., user input, file input, network data) can be internally tagged ($tainted$). This tag is carried on throughout the execution.

A sensible example for tainting are downloaded applications. At execution time, the application's input data are automatically tagged. When the application operates on tainted data (i.e., data originating from input tags) data read from those file descriptors will automatically inherit the taint, i.e. the tag is propagated along the flow of information during the execution of an application. Such propagation functions can be described within a framework, e.g. [LC06a]. A global table of tainted memory areas, often called shadow memory, is kept by the runtime environment and evaluated when memory is accessed. For all assignment operations, the new taint information (typically a bit-vector of taints) for the left-hand-operand is derived from the right hand operand's taint information using the inheritance or propagation function mentioned above. A tool that follows this approach is Policy Scope [ZJS+09].

> ... "Information flow tracking (IFT) refers to the ability to track how the result of a program's execution is related, via either data or control dependencies, to its inputs from the network, the file system and any other external parameters such as environment variables and command line arguments. To accurately track information flow, each piece of input should be assigned a tag, which could be a bit (e.g., a taint bit) or a pointer to an arbitrarily complex metadata structure, and for each assignment operation, the tag of the assignment operation's left-hand side is derived from the tags at its right-hand side according to certain tag combination rules. Different information flow tracking applications require different types of tag and use different tag combination rules." ...
>
> Lam and Chiueh in [LC06a].

Especially in mobile environments, where the necessity for access control for applications (e.g., access on private data such as phonebook and location in-

formation) has been recognized, a complementary solution would track *how* data are used at application level. The TaintDroid framework targets exactly this [EGC+10]. TaintDroid is a modification of the Android operating system (Linux-based) and runtime environment (DEX, Dalvik Execution Environment, an implementation of Java). It defines several taint sources, which contain privacy-related data, and follows the piece of information throughout the lifetime of an application. It uses a rule-based logic to propagate taint tags in memory and for execution. By regarding library-based code as trusted, they manage to achieve only little overhead, an average of 14% increased execution times. Figure 2.7 shows the propagation of tainted data in memory.

When binary code is not interpreted but directly executed, binary instrumentation can be employed in order to inject taint-tracking functionality into the execution. We discuss the existing approaches in the corresponding Chapter on page 94.

In contrast to interpreted or binary code instrumentation, there are also source-to-source frameworks, which translate given C code into runtime traceable C code. The frameworks GIFT [LC06b] and DIVA [XBS06] apply this technique. While this technique has the advantage of a limited overhead, a fundamental disadvantage is that the source code of programs needs to be available. As such, source-to-source information flow tracking cannot be applied to third party binary programs.

A hardware-based approach shows that taint processing does not need to be limited to interpreted or symbolically executed code. In [KDK09] a co-processor is used to follow tainted data of generic programs run on the host CPU.

Information Flow Monitoring in Distributed Environments A framework for information flow monitoring in distributed environments under the name "Aeolus" has been proposed in [Che09]. Its security model is based on information flow control and introduces a number of mechanisms that are applied to code that is run at an intermediate level (e.g., Java and .NET). Similarly, a system called "Laminar" also sets up on Java VMs and introduces labels [RPB+09]. Labels can be distinguished between security and clearance labels. The label concept is based on information classification, such as used by military information retrieval services. These frameworks target the confidentiality of certain information and documents. Their approach utilized middleware functionality, thus they do not follow indirect data flows inside the application, such as it can be done with binary instrumentation approaches. An approach that uses local binary instrumentation and instrumented TCP/IP connections to propagate the tag with so-called "Flow Managers" is presented in [KKCS09]. Recently, libdft [KPJK12] provides a toolbox for information flow tracking, which was used to distribute tags between processes and hosts [ZPK11].

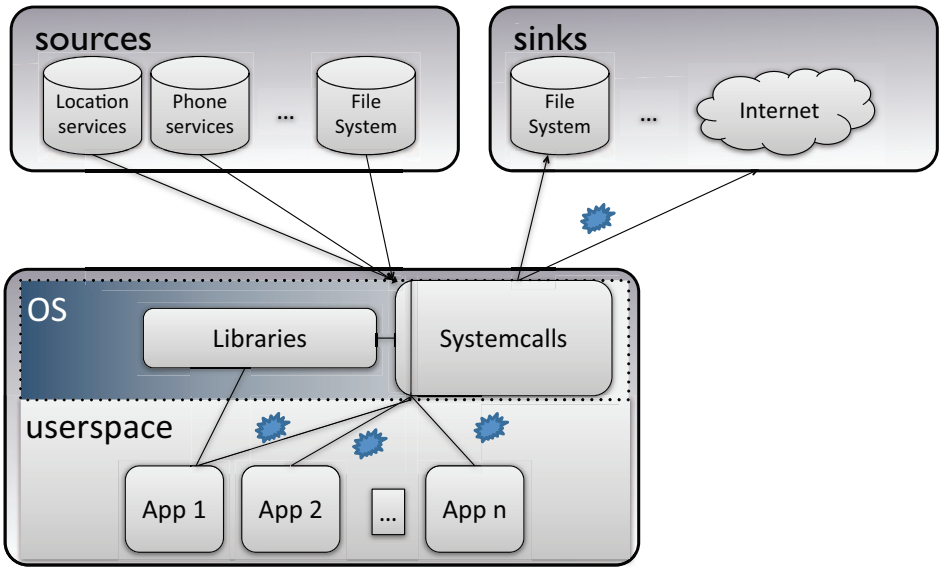


Figure 2.7: Propagation of tainted data (blue) in memory as implemented in TaintDroid. Data from taint sources is marked as soon as delivered by OS services (system calls, file system access). Libraries are regarded as trusted and thus aren't affected by the taint engine [EGC+10].

2.3 Dynamic Application Environments and Platform Security

In the following, we define *Open* and *Closed Application Environments* and categorize techniques presented above that are used to dynamically analyze security properties by analyzing data- and information flows at runtime as well as static methods to contain untrusted code.

2.3.1 Application Environments

Definition: An *Open Application Environment* is an environment in which the user controls, which program he wants to run. There are no measures taken to hinder the user from installing or running additional software.

Open Application Environments were the default for all personal computers until very recently. Introduced for the sake of security, application environments for higher level code such as Java, performed security checks (i.e., cryptographic verification of a signature) but always concerned the user as last instance to override security policies and allow the execution of program code even when

an invalid or self-signed certificate was presented[2]. Within operating systems, it has been a design principle since the 1960s (introduced with multics) to separate critical code (kernel and low level drivers) from user-space programs [Gol10]. This is now taken towards a finer granularity, i.e., user-space code is differently trusted based on origin, authenticity of signature and other methods. Thus, there has been a major shift towards *Controlled Application Environments*.

Definition: A *Controlled Application Environment* is based on the idea that only previously authenticated code may run on the platform. This approach is followed for various mobile devices, of which Apple's iPhone is the most prominent one. It uses a closed execution environment, includes a peer-review process for applications, as well as cryptographic signature and a closed distribution platform, the so-called AppStore).

A Controlled Application Environment can be enforced with multiple techniques. The most commonly used techniques are are:

1. Selection and Review of Software published
2. Code signing for programs, which passed the previous selection
3. Distribution Platform: Controlled form of distribution and revocation
4. Enforcement of code signatures on the target platform, using restricted execution, such as sandboxes.

We can see that in addition to the technical measures (code signing and sandboxing), non-technical policies are followed in the process of publishing an application.

Figure 2.8 shows a comparison of techniques used to achieve a secure execution of code and how they are applied in different environments.

A combination of techniques is often applied to achieve higher level security goals (for example the containment of privacy related data and possible attacks out for the code). The techniques explained are often complementary, as they cover different aspects.

Definition: A *Static Environment* is commonly used for embedded systems. No additional software can be installed without changing integral parts of the system (i.e., firmware updates, if maintenance is foreseen). Even though interfaces, especially for communication, are well-defined, also in embedded systems

[2]Which was a good idea, because the acquisition of signatures from a trusted party (PKI) was expensive for developers - but was a bad idea on the other hand, as it educated users to discard warnings about executing code from untrusted sources

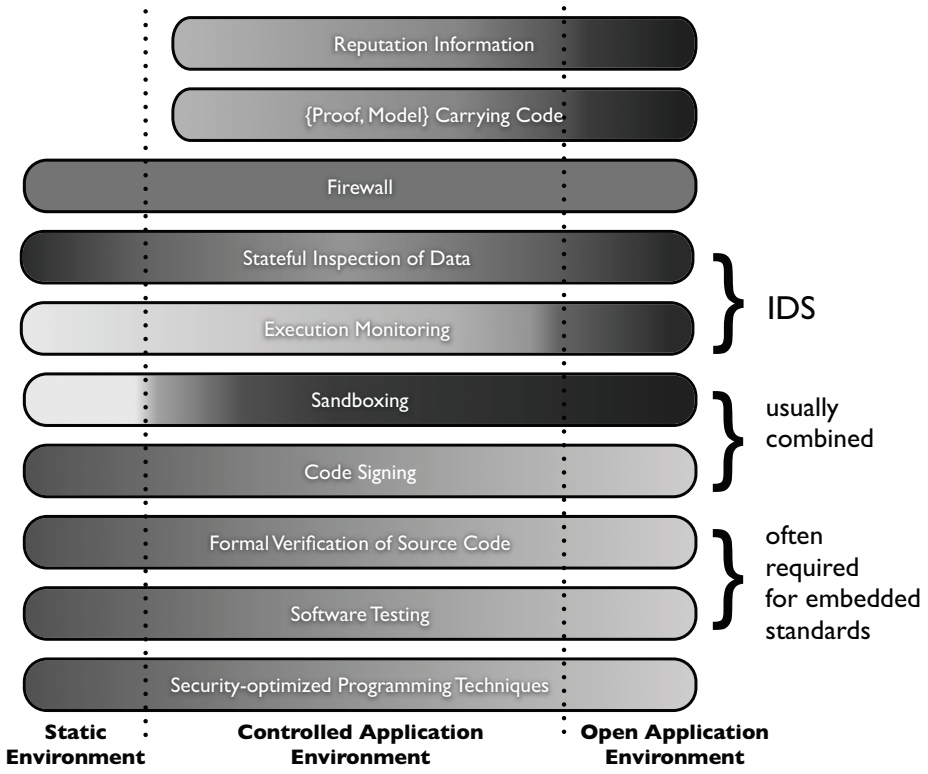


Figure 2.8: Classification of usability of security techniques used in different application areas. Darker areas show more usage (and applicability), while lighter areas show limited use or little added security. Static environments are only updatable but do not execute mobile code, while controlled and open environments can load additional software and code at runtime.

the trend goes towards dynamic behavior as envisaged by a variety of sensor networks.

Reputation based security models do not apply to static environments, as these do not allow foreign applications to be installed. The same applies to proof- and model carrying code. In contrast, these approaches are well suited for open application environments. In open environments, the $user$ of potentially unknown software may want to find out about certain behavior and/or experience from other users, referring to a knowledge base.

2.3.2 Example for Closed Application Environments: An App Store

To exemplify the combined use of security mechanisms, we show how security features are combined in order to achieve platform integrity. In Figure 2.9 it is shown how all of these measures are utilized by Apple in order to control their

iPhone platform, a famous example for rigorous policy enforcement in 2009 and 2010. In Table 2.1 we show details on the individual steps and classify these into technical and non-technical measures. Since its introduction in 2008, many other such stores have copied the concept (e.g., Amazon AppStore or Windows MarketPlace), while some deliberately do not require a review and certification process (Google Play, a.k.a. Market).

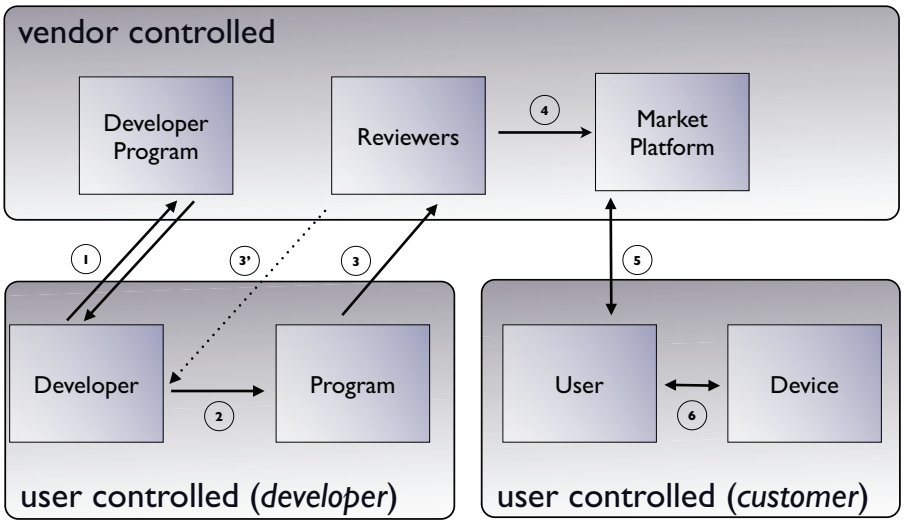


Figure 2.9: Enforcement of Policies and Security Measures: Apple's AppStore as Example. Steps 1 to 6 have to be taken before an application can be used on a user's device. The steps are: (1) Enrollment of Developer, (2), Software Development using obtained Credentials, (3) Submission to Review Process, (3') Rejection and Feedback from Review Process, (4) Code Re-Sign with Apple Credentials, (5) Download from Closed and Controlled Platform, (6) Sandboxed Execution in Controlled Environment along specially secured boot process.

Step	Description	Classification and Measure
1.	Registration as Developer	Non-Technical Policy
2.	Development: Keys and Certification	Technical: PKI and code-signing
3.	Submission for Review Process	Technical: PKI and code-signing
3'.	Feedback from Review Process	Non-Technical Policy
4.	Distribution via Platform	Technical: PKI and code-signing
5.	Download and Installation	Technical: closed platform (iTunes) and userbase (Apple-ID)
6.	Execution on Target Device	Technical: sandboxed environment, hard signature verification and securely booted device.

Table 2.1: Classification of Individual Steps in the App-Store Example. Graphical representation is found in Figure 2.9.

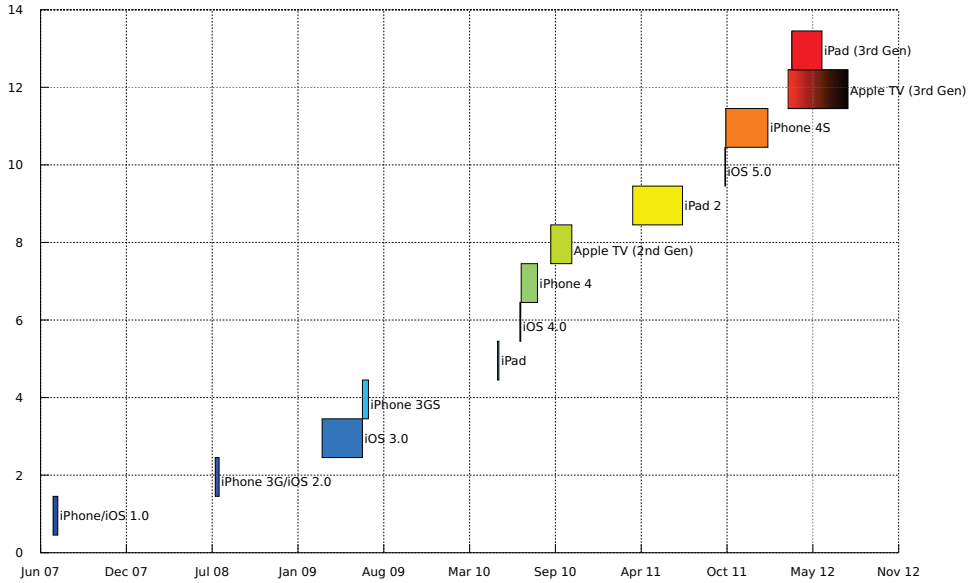


Figure 2.10: Apple iDevice and iOS Jailbreaks: The bars show the time between the release of the corresponding device or iOS version and the point in time when its security was first broken. The only yet unbroken device is the Apple TV (3rd Gen.). The dates of release and jailbreak are from the Wikipedia article "iOS jailbreaking".

2.3.3 Attacks on iOS Security

The security concept that Apple has applied for their iDevices (iPod, iPhone, iPad) is very stringent and ranges from a secure boot procedure up to the software certification process described above. Many more features complement the OS security [App12]. However, due to the fact that smartphones are widely deployed and promised an interesting target for hackers (i.e., those who are interested in free-to-use devices), these security measures were often broken only short time after the introduction of a new device and/or firmware version. We show a timeline of the evolution of this cat-and-mouse game in Figure 2.10. While some of the bugs that were exploited were hardware-based (e.g., flaw in baseband processor's boot process), others were purely software based and could be exploited at runtime.

2.3.4 Complexity and Cost of Approaches

In addition to Figure 2.8, where we showed the different applications of security techniques presented in this Section, we show how complexity and cost are differently distributed between the developer, the target platform and potentially required infrastructure. Figure 2.11 shows this distribution, in which we rated the efforts at different points on a five point scale (from $(--)$ to $(++)$ and n/a where not applicable).

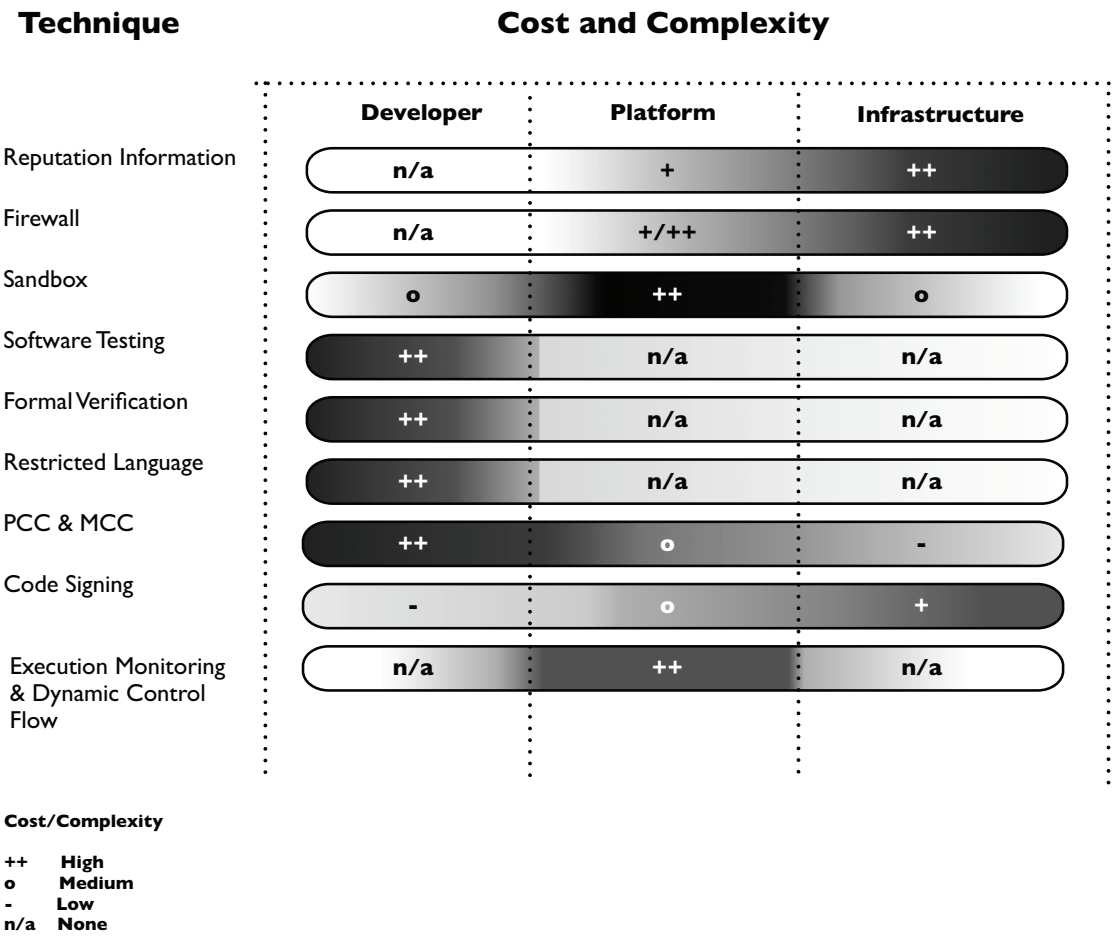


Figure 2.11: Depicted are the efforts necessary for different security techniques. The effort is shown on a black (high cost/complexity) to white (does not apply / none). We show the applicability of techniques at three places: During development, at the execution platform, and for the infrastructure surrounding the execution platform.

2.4 Conclusion

In today's computer systems, security is an integral part. Despite the fact that anti-virus programs and other intrusion detection solutions provide a "visible" and often efficient protection, the major requirement for a good security foundation is the underlying security concept. This is true for any kind of operating system, runtime environment, and also for network infrastructure.

This paradigm is well understood in the network and computer world. While computers and networks had not been well protected in the early days of the internet, as demonstrated in 1988 by Robert Morris Jr. with the first internet worm [Orm03], and similarly for personal computers that had no communication links [And02], a good security concept is now an integral part of every major operating system.

We see that the vehicle domain is currently changing towards opening itself for third-party software and device integration, creating the necessity to protect existing systems from attacks. While it is important to protect these systems, today security is rarely taken into account during specification and testing processes of control units and communication architecture.

Approaches exist to secure dedicated parts, such as immobilizers. However, an overall security concept for on-board networks and vehicle control units covering the complete vehicle infrastructure, has not yet been installed. Different domains are currently growing together in the vehicle environment, which makes the combined application of security concepts from those domains hard, as they rely on different techniques and measures, which are sometimes even incompatible. A comprehensive solution for this problem is still missing.

Current Automotive Security The security of the in-vehicle electronic architecture has so far mainly been maintained by isolation and secrecy of implementation details. The vehicle's networks themselves have not been physically accessible easily, as the car was a self-contained system with only some sensors connecting it to the environment. A physical connection to the wires of a bus is necessary to interact with the network's components, i.e., the ECUs. The principle of "isolation and obscurity" has been used for several decades in the automotive industry, for instance for immobilizers as mentioned above. While it seems beneficial in the first place, it can not substitute a sound security architecture in the long run, especially when the network architecture brings more and more remote interfaces into the car. The approach to keep implementation details secret is a very obvious breach of Kerckhoffs' 2nd principle "Security by Obscurity"[3]. The obscurity of bus protocols and CAN identifiers for instance

[3] Kerckhoffs' 2nd principle: The system must not rely on a secret, such that a lost secret will break the system. (original: «Il faut qu'il n'exige pas le secret, et qu'il puisse sans inconvénient tomber entre les mains de l'ennemi»), originally dedicated to crypto systems [Ker83].

is not given anymore today. Standard and transceiver chips are openly accessible and many CAN identifiers that still remain proprietary and confidential are published on the internet, or can be identified by reverse engineering. Thus, only the location of the cable (for physical access) remains secret – a secret on which the security foundations of a vehicle are built. Even more so, many devices that are connected to the internal network and expose, at the same time, a user-accessible interface that can even be compromised remotely under some circumstances.

We believe that an adaptation of the development paradigm is required to include security by design. This change in paradigm was only recently set in motion by the publication of security analyses on the connected vehicle.

Chapter 3

Environment

> "We can only see a short distance ahead, but we can see plenty there that needs to be done."
>
> Alan Turing

This Chapter introduces the scenarios used throughout the thesis, defines an attacker model, and describes the hardware and software setup that were part of the setting in which the thesis work was performed, and to which it contributed.

3.1 Scenarios

The effectiveness of our approaches is demonstrated on three different scenarios: One scenario involves Car2X communication ("Active Brake"), while the other two are centered around the vehicle's head unit (playing music and customizing vehicle parameters). In the following, we explain the scenarios in detail with regard to communication links and involved control units. We present the individual techniques developed in this thesis on these examples.

3.1.1 Scenario I: Active Brake (Car2X)

The first scenario includes a radio link for Car2X communication. We have explained the general nature of Car2X warning messages in Section 2.1.4, and take up the concept of DENM warnings in this scenario. The scenario itself involves two separate vehicles: a sending vehicle (SV) and a receiving vehicle (RV). The sending vehicle emits the DENM warning message, which is received by the following vehicle. This warning message indicates an emergency brake

Table 3.1: ECU Names used in the Active Brake Scenario. The ECU names are prefixed with **S-** and **R-** for the **s**ending and **r**eceiving vehicle, respectively.

ECU	Description
BS	Brake Sensor
BA	Brake Actuator
BC	Brake Controller
CU	Communication Unit
IC	Instrument Cluster
AU	Application Unit

situation at the sending vehicle: an Emergency Stop Signal (ESS) is defined as "*a high retardation force has been applied to the vehicle*" [Uni10], that is, the brakes are engaged to decelerate by $a \geq 6\frac{m}{s^2}$ [Uni11]. The receiving vehicle may act accordingly, for example by displaying a warning message for the driver and by preparing for an emergency brake (e.g., to pre-charge the brakes, to engage belt pretensioners, and close the windows). In addition, if the confidence in the received message content is high enough, the brakes may autonomously be engaged. Plausibility checks that confirm the validity of data are part of the evaluation. Such plausibility checks may include the continuity of a vehicle's trajectory by collecting vehicle's CAM beacons and computing a local dynamic map as foreseen in ETSI, or by the support of additional local sensors such as radar, if the situation allows.

We show how individual ECUs are distributed in both vehicles in Figures 3.2 and 3.3. The ECUs are interconnected with CAN and Ethernet buses, over which information are exchanged. The communication between different units in the Active Brake system is referred to as steps $AB_{\{r,s\}\{1,...,4\}}$ for the receiving (r) and for the sending (s) vehicle and $(1, ..., 4)$ for the respective message. The message sequence is depicted in the chart in Figure 3.2.

In Figure 3.3 we show the two vehicles, along with primary ECUs that participate in this scenario.

In both vehicles, the Application Unit AU is used for key distribution as explained in the following Chapter, and is thus indirectly part of the communication chain of Figures 3.2 and 3.3.

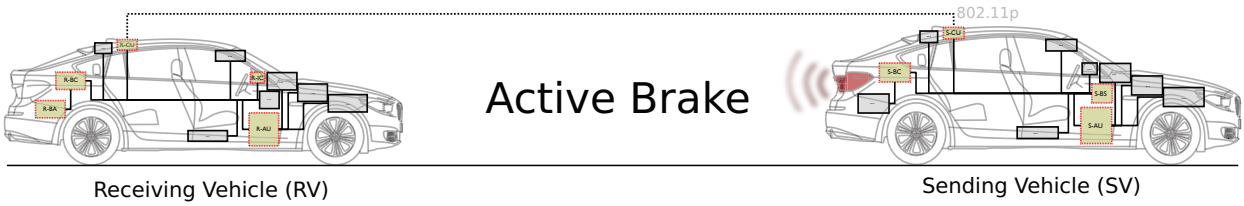


Figure 3.1: Active Brake: An emergency brake is performed at SV (the sending vehicle) and received at RV (the receiving vehicle). Participating ECUs are marked yellow.

Sending Vehicle (SV) The sending vehicle detects an emergency brake situation, in which the vehicle will decelerate with more than 6 m·s^{-2}. The initial brake request is generated by the Brake Sensor (S-BS) inside the vehicle's pedal. It is then transmitted (AB$_{S1}$) to the Brake Controller (S-BC) and evaluated there for plausibility. The emergency signal is then generated and distributed (AB$_{S2}$) internally (not shown) and to the Communication Unit (S-CU). The emergency brake message is broadcasted (AB$_{S3}$) as a DENM wireless message via IEEE 802.11p to surrounding vehicles, notably RV, by the communication unit.

Receiving Vehicle (RV) The following vehicle, RV, receives the message (AB$_{R1}$). The received DENM is checked with regard to its authenticity (valid message signature) and authorization (a valid Car2X pseudonym certificate) at the communication unit. The messages (AB$_{R2}$ and AB$_{R3}$) are then internally distributed to the Brake Controller (R-BC) and the instrument cluster (R-IC). The instrument cluster display will show a warning to the driver. A last plausibility check at the brake controller is performed before the brakes are engaged by sending a message (AB$_{R4}$) to the Brake Actuators (R-BA).

In our demonstrator prototype the whole communication chain was secured and realized as part of two BMW research vehicles, with the exception of physically engaging the brakes, which we did not demonstrate. See Annex B.6 for pictures of the prototype.

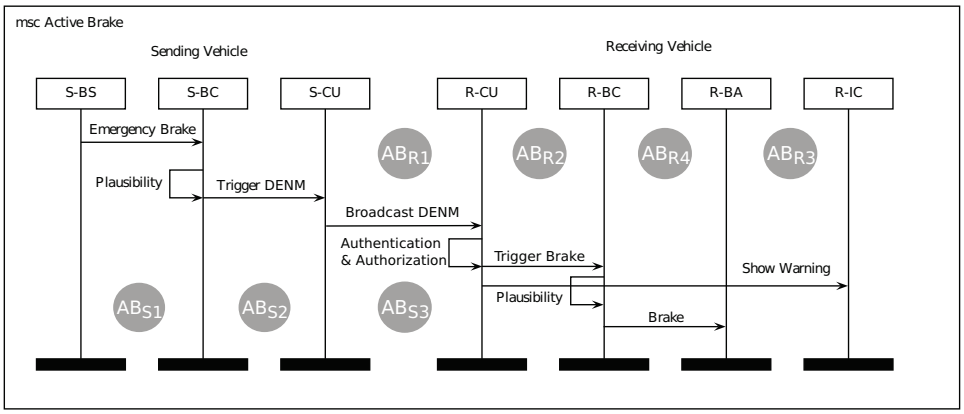


Figure 3.2: Message Sequence Chart for Active Brake Scenario: The emergency brake signal is given in the **S**ending vehicle and received and evaluated in the **R**eceiving vehicle. At intermediate steps, the data is checked for plausibility. We used the sim^{TD} Communication Unit to transmit DENM notifications between the vehicles. The ECUs are abbreviated according to Table 3.1.

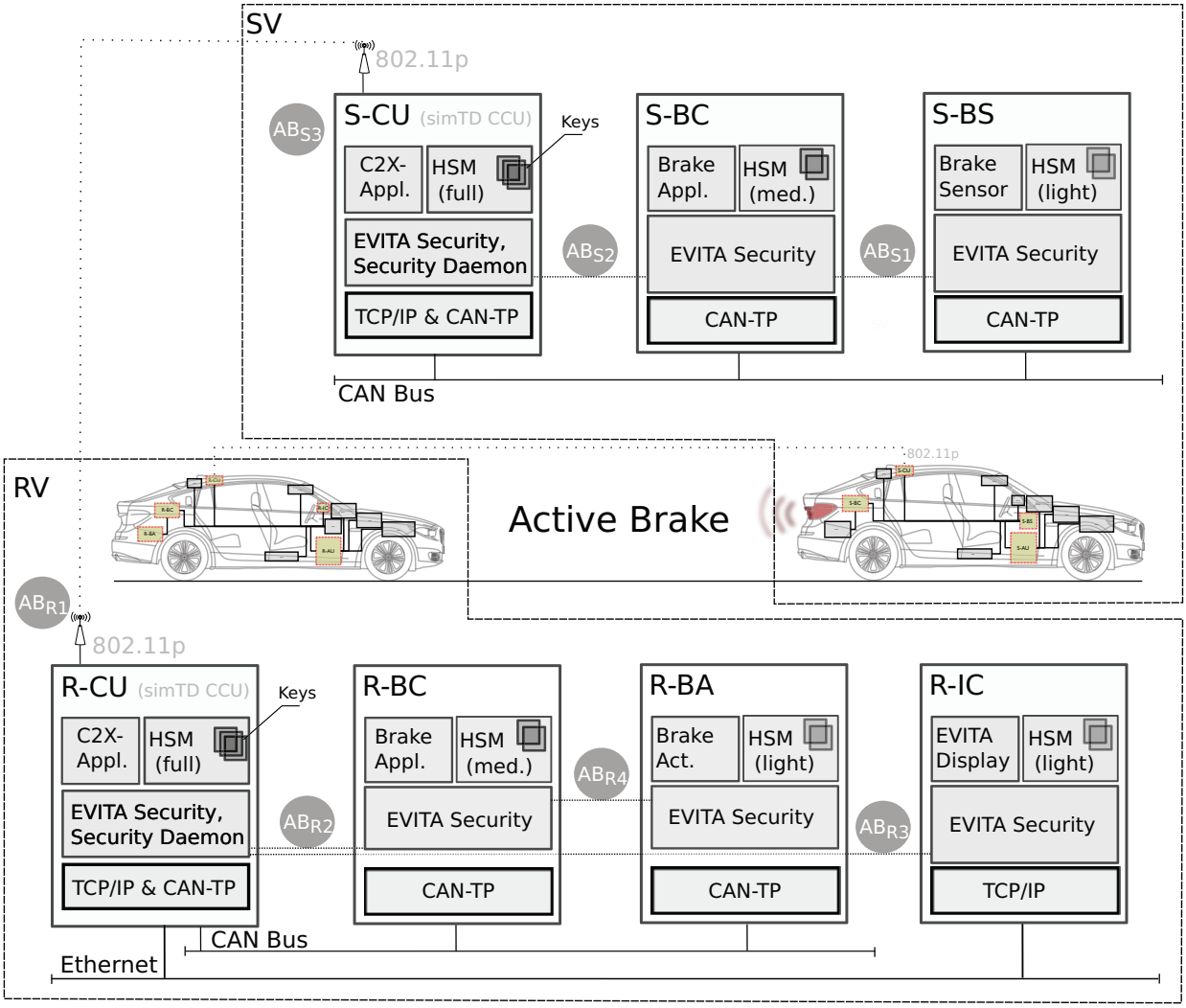


Figure 3.3: Components of both vehicles: An emergency brake is performed at SV (sending vehicle) and broadcasted to other vehicles, including RV (receiving vehicle) via 802.11p, i.e., Car2X communication. Internally, several ECUs participate in the generation (SV) and evaluation (RV) of this message. The participating ECUs are marked in the vehicle schematic.

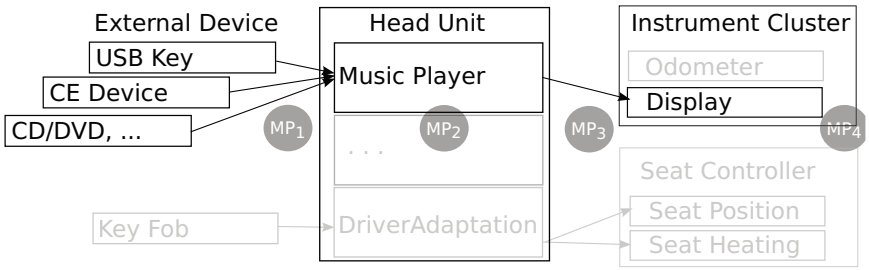


Figure 3.4: Scenario II: Playing Music on the Head Unit. The music's metadata is extracted and sent to the Instrument Cluster, where it is displayed to the driver.

3.1.2 Scenario II: Playing Music

Passengers listen to digital recordings in the vehicle since the Compact Disc has ruled out cassettes. However, nowadays digital recordings can be played in various formats, e.g., MP3, WMA, M4A, and from various sources like CDs, USB thumb drives, SD cards, or wirelessly from a mobile phone. These files of different formats already provide a large attack surface that can be used to compromise the head unit, as successfully shown with specially crafted WMA files in [CMK+11]. Meta-information such as the artist and title are extracted from the music files themselves. Both the parsing of the file for meta data and the actual music decoding are done on the head unit. The metadata are passed on to the vehicle's instrument cluster, i.e., the display adjacent to the speedometer, where the song title and artist are displayed to the driver. If one of the processing software entities does not sufficiently validate the input and programming flaws are present, an attack may use music files to inject malicious code.

This scenario addresses playing music from an untrusted source, e.g., a file on disk, and disseminating the information throughout the on-board network.

In Figure 3.4, the involved entities, i.e., the media source, head unit, and instrument cluster are displayed. The media file itself is locally read from a file via USB, or a proprietary device interface, or wirelessly accessed via Bluetooth or WiFi from a mobile device (MP_1). It is played back locally at the head unit (MP_2) and metadata are extracted and submitted to the instrument cluster (MP_3), where they are displayed (MP_4).

3.1.3 Scenario III: Driver Adaptation

Vehicles with power seats are often equipped with up to a dozen DC-motors to adjust the seat's position. They also offer the possibility to store the current seat position in memory. Some vehicles combine this with the identification of a specific key fob. In our third scenario, an application on the head unit receives

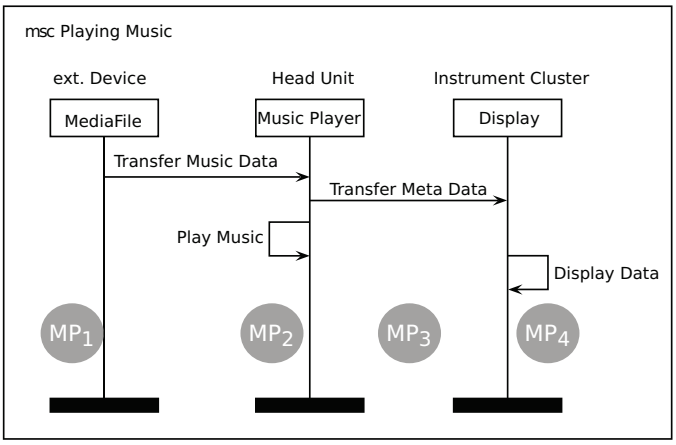


Figure 3.5: Message Sequence Chart for Scenario II.

information about which key fob is used (i.e., its identifier) and adjusts the seat position according to a saved profile (on disk). In reality, this step would usually involve at least one more intermediate component, which controls the radio communication with the key. Also, in order to authenticate the key fob and enforce distance bounding, the communication would require more than one message.

The key fob's identifier is read wirelessly, e.g., via standard Radio-Frequency Identification (RFID) (DA_1). A database with stored driver preferences for this identifier is read (DA_2) on the head unit. According to these preferences, the desired seat position is sent to the seat controller (DA_3), where several servos are triggered to arrange the seat (DA_4). The setup and a sequence chart are depicted in Figures 3.6 and 3.7.

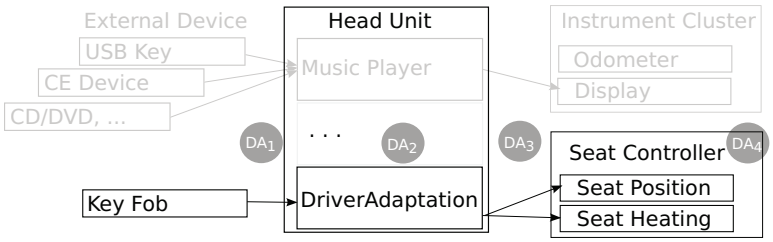


Figure 3.6: Scenario III: Driver Adaptation. Seat positions are automatically adjusted depending on the key-fob used to unlock/operate the vehicle.

3.2 Attacker Model

There are a number of motivations for attackers of modern vehicles. The number one purpose to break into a vehicle is stealing the car itself. While this is

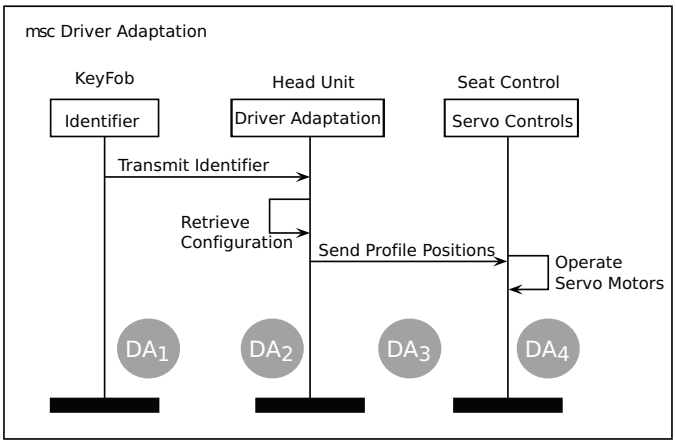


Figure 3.7: Message Sequence Chart for Driver Adaptation: An identifier is retrieved wirelessly from an electronic key (simplified with only one message). According to a local profile the seat positions are adjusted.

definitely a realistic threat to vehicle security, our attacker model anticipates more sophisticated attacks against the vehicle. Such attacks may originate from the vehicle owner himself, i.e., in order to tweak and tune the vehicle's functionality such as:

- Changing engine controls for increased performance,
- Tampering with the vehicle's mileage,
- Removing technical restrictions (e.g., watching TV while the car is moving),
- Unlocking features (e.g., navigation system),
- Adding functionality (e.g., compiling and running own applications, changing the navigation system).

While some actions, like changing the radio/navigation system may be benign, others influence the vehicle safety (watching television while driving) or create financial loss (unlocking paid functions, changing the mileage).

As the vehicle is a safety critical domain, the manufacturers usually deny liability once the closed system is altered (e.g., an additional device is added to the vehicle's network, which may perturbate the normal operation of safety systems). Due to the enormous demand for dynamic content and continuously enhanced consumer experience, manufacturers are forced to allow connected and updatable content in close interaction with their in-vehicle infotainment systems. By this, the vehicle exposes several interfaces, ranging from consumer-oriented USB or Bluetooth access to back-end oriented 3G connections. This creates a large additional attack surface in comparison with existing closed-system approaches.

For our scenarios, we do not differentiate between internal and external attackers, but assume that an attacker can have physical access to the vehicle, at least for a short period. This means that the attacker can both read data from internal networks and inject or replay data. The attack itself can take place at a later stage, e.g., by using obtained or injected data or code remotely. A non-physical attack offers similar opportunities if the attacker can freely access network interfaces after compromising an ECU.

Definition: The *attacker model* is defined by the following abilities and constraints. An attacker can access all communication interfaces of the vehicle: it may read data on all buses and interfaces and write data to all interfaces. The attacker can upload certain programs or replace ECUs of the vehicle, but cannot access data internal to HSMs that are part of the ECUs. We assume that the combined CPU/HSM package is tamper resistant and no invasive attack, such as micro-probing or focused ion-beam attacks (e.g., in order to zero memory or tamper microcontroller fuses) is made. Lessons regarding these attacks have been learned in the smart-card industry [KK99] and apply to this area similarly.

Our attacker definition covers internal attackers with unlimited time, such as the vehicle owner, as well as external attackers who can gain short term access to the vehicle (through social engineering, physical force or implementation flaws at exposed interfaces, such as wireless networks).

3.3 Possible Attack Vectors

We have selected these three scenarios, as they cover most of the vehicle's common communication patterns as well as all of our attack contexts. In the first scenario, modifications of the Car2X payload, i.e., $AB_{\{S3,R1\}}$ itself are out of the scope of this thesis. Nevertheless, an integration of Car2X communication was done in the scope of the EVITA project demonstrator with an existing communication unit from the sim^{TD} project. This link is secured by ECC signatures using pseudonym certificates as foreseen for standardizations. Modifications of radio messages can thus easily be detected by the CCU and would be discarded.

Injecting bogus payload to openly accessible interfaces poses a high risk in all of the scenarios. A primed music file or an invalid keyfob-identifier may be used to inject software exploits on ECUs at runtime, as shown in [CMK+11].

Once an attacker has had access to the inside of the vehicle (physically, or by exploiting an exposed interface such as Car2X, Bluetooth, or others), the attacker may currently replay and falsify any data that is exchanged via onboard networks. None of the currently used networks (LIN, CAN, FlexRay, MOST) authenticates or encrypts the payload. This is done at the application

level and only for very specific functions, such as updating an ECU's firmware in diagnostic mode.

As part of the EVITA project a study about attack risks and security requirements was conducted [RWW+09]. It includes a catalogue of over one hundred different requirements covering privacy, authenticity, and confidentiality for different automotive applications. These were combined with the risk and the attack potential, i.e., the likelihood of a security-related incident and the severity of a successfully mounted attack. One of the outcomes of the study was that most currently existing functions and communication signals can be well protected with elementary authentication, as the attack risk is low. On the other hand, exposed interfaces and execution platforms pose a major risk, where stronger security functions need to be applied. This inequality of required security levels is addressed in this thesis in Chapters 4 and 5.

Specific attacks that target network communication or exploiting applications at runtime and how they would be applied to the three scenarios, are explained in the following.

3.3.1 Attacks: Network Communication

Within an unauthenticated broadcast network, every node may read traffic and send traffic under any identity. This is true for classic IP-based networks by forging the source address, and also for vehicle networks by forging the CAN identifier. This class of attacks is called *spoofing*. Within our scenario, all communication links are affected if the payload is not secured.

Another basic kind of attacks is to *inhibit or drop communication*. Dropping network frames can only be done at relaying nodes, i.e., gateways. This requires the gateway itself to be compromised. In contrast, inhibiting genuine communication can be performed on any network node connected to a broadcast medium, simply by provoking collisions during ongoing communication. A modified transceiver chip would be necessary to perform such a kind of attack, as media access is usually handled in hardware.

An often overlooked technique on authenticated or encrypted communication are *replay attacks*. If the attacker is able to intercept and inject messages, a genuine message can be replayed at a later time. Sequence numbers or timestamps provide are countermeasures against this attack, yet require a secure and authentic time to be distributed beforehand.

Man-in-the-middle attacks are a technically more sophisticated form of intrusion. The original signal is received and dropped, and a modified signal is injected. Under certain circumstances like a compromised certification infrastructure, this attack even works for authenticated and encrypted communication. For IP-based networks there exist tools to redirect traffic by ARP-spoofing

and to hijack the communication. Within the vehicle, devices for a physical man-in-the-middle setup are being sold (variants for CAN and MOST buses exist). They transmit wrong velocity signals to the head unit in order to enable TV output during driving operation, despite the exclusive policy implemented by the manufacturer.

3.3.2 Attacks: Host Based Intrusions

In order to obtain data, or to access resources such as the I/O channels of a device, several intrusion types can lead to success. The *modification of the ECU firmware* or of the bootloader in memory can make it possible to inject malicious programs. This kind of attack requires physical access to the target system.

The class of *runtime attacks* is much larger: Specially crafted input is used to divert the control flow of applications. Once injected or reconstructed code is executed, the attacker has gained the same access level as that of the program that is *exploited*, i.e., a vulnerability is actively used to gain access. If system resources are not specifically protected, as it would be the case for today's multi-user systems, an attacker can take any kind of action. This involves intrusive attacks such as distracting the driver by influencing the vehicle behavior and showing warning messages, but also stealthier behaviors, such as recording audio from connected microphones or tracking the vehicle's location.

The class of runtime attacks is of special importance as they can potentially be performed remotely. Techniques and countermeasures used for attacks related to overwriting data on the stack, such as the return address or function pointers, are explained in Section 5.4.1 (p. 93).

3.4 Towards A Secure In-Car Architecture

This Section describes the in-vehicle hardware and software architecture of the EVITA FP7 European Project. This thesis was written in parallel to work in the EVITA project, so a number of contributions that go beyond what is described in the thesis, were made to the architecture, protocol design and implementation.

This architecture provides a foundation for experiments and higher level solutions developed within this thesis. As part of the EVITA project, an analysis of attack scenarios and security requirements [RWW+09] with regard to typical automotive use cases [KFL+09] has been done. The security architecture is a combination of software and hardware elements [WWZ+10]. While the software manages security functionality and provides automotive applications with security services, such as authenticated or confidential communication, the hardware part provides the trust anchor: a trusted platform for storing key material and

using cryptographic primitives. This hardware/software co-design assures that cryptographic keys are i) not used unauthorized, i.e., the platform needs to be securely booted for the key to be used, and it can only be employed according to its use-flags, and ii) will never leave the hardware in an unencrypted form, i.e., all cryptographic operations take place in the hardware module. Note: Despite secure boot and key usage control, software that accesses an HSM can still be exploited at runtime.

3.4.1 Software Security: The Framework

One of the basic building blocks that was used in order to implement and test protocols for the automotive on-board network was the Embedded Vehicular securitY (EMVY) framework provided by our research partner BMW. This framework provides the building blocks needed for a flexible security architecture. Applications use the services provided by the framework, such as "single sign on" or the establishment of secure communication channels.

The EMVY deployment architecture is constructed in a master-client manner. So-called *master nodes*, which provide services to clients, are deployed on the on-board network and communicate with clients through a Server-Remote Procedure Call (RPC) interface. Similarly, the Clients can be reached over a Client-RPC interface. These interface uses ASN.1 encoded request and response packets.

Using RPCs on the server allows to use lightweight implementations at ECUs with limited processing power and memory. In particular, the client nodes were implemented in plain C, which makes them suitable also for low-performance nodes in the vehicle, such as low cost sensors.

In contrast to this, services that reside on master nodes are implemented in C++. They provide more complex functionality.

The prototype implementations that are presented in this thesis have contributed to the E-Safety Vehicle Intrusion proTected Applications (EVITA) EMVY framework and were integrated in the scope of the project. More technical details on this architecture can be found in the EVITA deliverables documents D3.2 [WWZ+10] and D3.3 [SIR+10] that can be retrieved via the project's website [EVI11]. An overview of the architecture is published in [AEKH+10]. The EVITA software architecture EMVY is shown in Figure 3.8.

3.4.2 Communication

The EMVY communication is encapsulated in a module referred to as CCM, the Communication Control Module. On top of socket and datagram communication, it can assure node authenticity in combination with the Entity

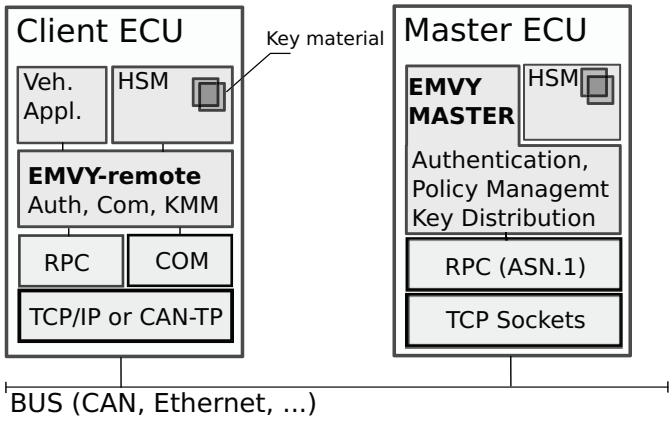


Figure 3.8: Overall EVITA Software Architecture: The EMVY model. A Master node (implemented in C++) provides high-level services such as authentication, policy decision, or key distribution, while low-end nodes (clients, using the "EMVY-remote" C library) connect to these services via an authenticated RPC interface. The HSM is part of the architecture and used via the EMVY application model.

Authentication Module or confidentiality of certain communication by using cryptographic primitives of the HSM. If a secure communication channel is requested, the initialization-protocols implemented inside CCM will establish the necessary credentials, thus allowing applications to communicate without deeper understanding of the underlying mechanisms, similar to SSL, for example.

An application requests a secure communication channel with another entity (which may be a group) and a certain security property set. This set is a subset of {`authenticity`, `integrity`, `confidentiality`, `non-replay-ability`}, with subsequent levels, e.g., to establish a certain trust by using keys or authentication codes of a certain length. The channel is established by triggering a key exchange, which we present in Chapter 4. After a successful key distribution, the channel can be used by the application and data is automatically wrapped with the selected security mechanisms (e.g., timestamps, encryption, or Message Authentication Codes (MACs), and even fractions of MACs).

3.4.3 Policy Decision

One of the core modules of the framework is the policy engine. Policies are defined in a common `S,P,O` format: A subject `S` may interact with object `O` using predicate `P`. For example, a process (the subject) accessing files on a hard disk (the object) would require policies to allow read or write operations (the predicate).

In the framework, policy decisions are made on the master node and enforced on client nodes. Thus, authentic and timely communication between nodes

is crucial. Decision may have a validity timeframe, during which they can be cached at the enforcement points (i.e., the clients).

3.5 Hardware: The Hardware Security Modules

One key aspect in the EVITA architecture is the Hardware Security Module. It stores cryptographic key material and processes all cryptographic operations, such that credentials that must be kept confidential at all times, *will never leave this module unencrypted* and will similarly never be loaded into the application CPU's registers or memory. Only a handle to these cryptographic keys can be obtained by the application during an HSM session.

3.5.1 Key Usage

The usage of individual keys can be controlled by multiple so called *use-flags*. These flags provide a low level access and usage control. In particular they bind the usage of this key to certain configurations, certain functions, or to means of separate authentication, such as a password. These conditions must be met in order to use a key with the HSM.

Secure Boot Flag The most prominent flag is the secure boot flag: It binds a key to a successful boot procedure with known hash values in the ECU Configuration Registers (ECR). These registers work similar to a TPM's PCR register, i.e., measurements are taken at each stage of the boot process. Binding a key to ECR measurements makes sure that the key is not used by a replaced firmware, and only in the intended and correctly booted environment. The secure boot process was successfully demonstrated on an Infineon TC1797 microcontroller loading AUTOSAR. We require all the keys that are part of the protocols that we introduce in Chapter 4 to be bound to a securely booted ECU.

Operation Specific Flags A key may be restricted to certain operations: It may, for example, only be used to verify cryptographic signatures, while the generation is forbidden. We built our key distribution mechanism around this feature in order to allow for low-cost asymmetric usage of symmetric key material.

3.5.2 The HSM programming interface

The programming interface of the HSM is kept as generic a possible. While underlying algorithms may be complex and differ fundamentally, for example

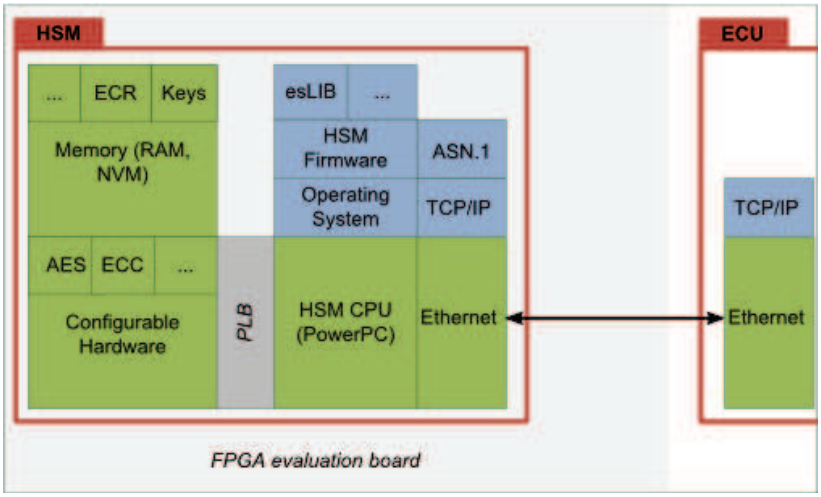


Figure 3.9: HSM Hardware Prototype Architecture: Cryptographic functions are deployed in hardware (green) on the FPGA board. The software (blue), i.e., HSM firmware control the HSM functionality and makes it available via a network API (Ethernet, ASN.1 encoded requests). Drawing adapted from [PGW+11].

asymmetric signatures vs. message authentication codes, the programming interface to this functionality is designed straight-forward. It comprises a number of core functionalities: key generation, key import and export (using transport encryption) and cryptographic operations (i.e., encryption and decryption in addition to signature generation and verification).

3.5.3 Prototype Platform

The HSM is implemented as part of a Xilinx Virtex-5 Field Programmable Gate Array (FPGA). Its cryptographic operations are realized inside the FPGA itself. All I/O operations, the key-storage, and the actual firmware are implemented as part of an embedded Linux (v2.6.34), which uses the Virtex440-PowerPC core and 256MB RAM available on that platform. As a prototype, the HSM itself is connected to the application CPU via Ethernet. An embedded SPI interface also exists but was used only in the context of an AUTOSAR proof of concept integration, which is not further discussed. The Ethernet/IP interface uses standard TCP sockets in order to exchange HSM commands that are encoded in a ASN.1 request/response packets. The architecture of the HSM prototype platform is shown in Figure 3.9.

3.5.4 Performance

Compared with software-solutions, the HSM offers a large improvement in performance: While in software only about 15 ECC-256 signature verifications are

feasible on a typical 400 MHz CPU, we achieve between 500 and 700 verifications per second using hardware acceleration. Current traffic estimations from the C2CC Consortium demand for at least 500 verifications per second for intersection scenarios, during which every car will send beacons at a maximum rate of 10 Hz, depending on its velocity.

Similarly, the performance for symmetric operations reaches data rates between 60 and 75 MBit/s for AES-128 in ECB and CBC mode. This is largely sufficient for all buses that are used in the vehicle today.

In Section B.1 we have analyzed how the overhead of the prototype relates to the actual processing time of the HSM.

3.5.5 Comparison with other Secure Hardware

The hardware security module was designed in EVITA in order to comply with automotive requirements: Cost effectiveness, tamper protection and specific variants were taken into account for the actual design of the prototype. As cost is the driving factor for automotive components, three variants of the HSM were specified. While the 'light' module can be produced at low cost, due to reduced functionality and thus chip surface, the 'full' module offers high-performance must be powerful enough to handle hundreds of signature verifications per second, as pointed out in estimations [G11]. A 'medium' variant provides asymmetric operations, but no ECC hardware acceleration.

A common Trusted Platform Module that is used in the PC is overly complex and costly compared with the 'light' HSM that is used to secure sensors in the vehicle. Likewise, for the specialized 'full' module, ECC cryptography is used, as this is foreseen for use with Car2X communication. Elliptic curve algorithms are not used in standard TPMs, which use RSA as asymmetric algorithm. While RSA is used in some software-only implementations at FOTs (sim^{TD} with 512-bit pseudonym certificates), it is not suited for wide deployment due to overhead through certificate and key length.

In the EVITA project, the HSM approach was compared to other hardware solutions by Wolf and Gendrullis [WG11]. In particular, the EVITA HSM is compared to TPMs as well as commonly used smart cards (SmC) and a specific automotive solution, the Secure Hardware Extensions (SHE) module [HIS09], specified by the Hersteller Initiative Software (HIS) consortium. We show their comparison in Table 3.2.

Long lifetime is an important characteristic of vehicles. While most electronic devices are only used for a few years, vehicles have a predicted lifecycle of over 15 years. Looking back 15 years, the MD5 algorithm was considered rather secure—today we have just seen how malware successfully uploaded malicious code via the MS Windows Update Service using a MD5 collusion in an old, but

still valid certificate[1]. These diametric requirements (fresh algorithms vs. long vehicle lifetime) pose a real problem. In the prototype implementation of the HSM generic point arithmetic operations were implemented, so that algorithms may be updated to a certain extent.

	Full	Medium	Light	SHE	TPM	SmC
Cryptographic algorithms						
ECC/RSA	■/■	■/■	□/□	□/□	□/■	⊞/⊞
AES/DES	■/⊞	■/⊞	■/□	■/□	□/□	⊞/⊞
WHIRLPOOL/SHA	■/■	■/■	□/□	□/□	□/■	⊞/⊞
Hardware acceleration						
ECC/RSA	■/□	□/□	□/□	□/□	□/□	□/□
AES/DES	■/□	■/□	■/□	■/□	□/□	□/□
WHIRLPOOL/SHA	■/□	□/□	□/□	□/□	□/□	□/□
Security features						
Secure/authenticated boot	■/■	■/■	⊞/⊞	■/□	□/■	□/□
Key AC per use/bootstrap	■/■	■/■	■/⊞	□/■	⊞/■	□/□
PRNG with TRNG seed	■	■	■	■	■	■
Monotonic counters 32/64 bit	■/■	■/■	□/□	□/□	■/□	□/□
Tick/UTC-synced clock	■/■	■/■	■/■	□/□	□/□	□/□
Internal processing						
Programmable/preset CPU	■/⊞	■/⊞	□/⊞	□/■	□/■	⊞/⊞
Internal V/NV (key) memory	■/■	■/■	⊞/⊞	■/■	■/□	■/□
Asynchronous/parallel IF	■/⊞	■/□	■/□	■/□	□/□	□/□

Annotation: ■ = available, □ = not available, ⊞ = partly or optionally available

Table 3.2: Comparison between different EVITA Hardware Security Modules, TPMs from the PC world, the SHE module for automotive, and Smart Cards (SmC) as generic crypto-in-chip device. Table from [WG11].

[1] Meant is the "Flame Spyware", which is attributed to intelligence services that developed the software for targeted operations.

Chapter 4

Communication Security

> "Never underestimate the attention, risk, money and time that an opponent will put into reading traffic."
>
> Robert Morris Sr.

In today's automobile, communication between different units is paramount. No vehicle would be able to ignite or drive without at least the core electronic components (e.g., engine control and brake systems) successfully communicating over internal networks.

As the vehicle's systems are more easily accessible, e.g., by connecting consumer devices via USB or bluetooth, but also with the introduction of Ethernet in current architectures, the on-board network becomes an easier target for attackers compared to classic vehicles without electronic networks.

For future cooperative Car2X applications, which are under standardization right now [Eur09b], the trust in received data is highly important. In current approaches, only the wireless link between vehicles, i.e., Car2X with IEEE 802.11p, is secured with cryptographic signatures using an adapted IEEE 1609.2 format [BSM+09]. The internal on-board network, although equally important in order to guarantee end-to-end security, has not been secured adequately.

In this Chapter, we take up this necessity to secure the communication between individual units of the on-board network. As contribution of this Chapter, we introduce i) a key distribution protocol and ii) a secure transport protocol for automotive networks. The communication protocols are designed with limitations of the communication bus and performance constraints of the computing architecture in vehicular systems in mind.

We designed the protocols and interfaces of this protocol in a way that they integrate, on the one hand, with Hardware Security Modules (HSMs) designed

in EVITA, and on the other hand with the software security framework EMVY as a prototype implementation. The integration of security functionality into software frameworks is important, as the architecture and functional design process of a vehicle's network is largely model driven. In a model-driven approach as taken in AUTOSAR, the framework assures that software functions can be flexibly deployed anywhere in the network.

As communication is a central building block for nearly all automotive functions that span among multiple ECUs, the techniques introduced in this Chapter apply to all of our three scenarios.

4.1 Key Distribution in Embedded Environments

One of the main building blocks for secure embedded communication is *key management*. It is essential to use fresh key material that is re-distributed periodically. If only static keys were to be used, attacks against these (never changing) keys would break the system for the rest of its lifetime. We use static long term keys only for the purpose of transport encryption and authenticating data, i.e., they are never directly used to encrypt or authenticate data apart from freshly generated session keys and will never leave the corresponding HSM apart from the initial pairing process that is done at the manufacturer (i.e., in a trusted environment).

The presented approach makes use of hardware security, in order to leverage performance by symmetric cryptography, while maintaining appropriate access control on the key material itself through the HSM as secure element.

4.1.1 Asymmetric Usage of Symmetric Keys

We use shared secrets (i.e., symmetric keys) due to constraints for cost and embedded deployment. Deploying those keys requires a small hardware component, the HSM at each ECU. The HSM implements accelerated cryptographic primitives and also offers secure key storage, thereby preventing any ECU compromise to spread to our key infrastructure. The HSM also mediates the access to cryptographic keys. The keys possess of additional data fields, where an *expiration date*, an *authentication code* or so-called *use-flags* are maintained. The use-flags enable the HSM to distinguish for which functionality a specific key can or cannot be used. Typically, the HSM will enforce some usage control on the symmetric keys it stores, based on those use-flags: for instance, a symmetric key may be tagged for *sign*ing (i.e., generating a MAC) at a single HSM while the exact same key resides at several other HSMs, where it is tagged for *verify*ing only. This usage control makes it possible to enforce an

asymmetric use of key material, while the computation itself is efficiently based on symmetric cryptography.

Definition: The key $k_{n,g}$ is the session key used for **g**enerating MACs at ECU_n (the sending entity) and the key $k_{l,v}$ is for **v**erifying the received data (including MACs) at ECU_l (the receiving entity).

Two other use-flags are used throughout this Chapter: the flags for $export$ and $transport$. The former allows a key to be exported in encrypted form and the latter allows it to be used to encrypt another key for export (that other key must be exportable). A cryptographic key can under no circumstances leave the HSM unencrypted. Of the session keys, the verification part ($k_{x,v}$ for ECU_x) must be marked exportable for key distribution. Additionally, a separate authentication key is used to validate an exported and transport-encrypted key.

For every ECU_n that establishes a secure channel with other ECUs, there are a transport encryption key $k_{n,t}$ and an authentication key $k_{n,a}$ deployed at its own HSM and the HSM of the Key Master ECU.

The EVITA project differentiates between three different types of HSMs, as explained in Section 3.5. While full and medium HSMs support and accelerate asymmetric cryptography, we focus on the use of the light HSM, which provides a hardware based implementation of symmetric cryptography primitives only. This type of module is most interesting for integration into low-cost sensors and actuators, as symmetric cryptography is less expensive in terms of gate count [PLSP07, Kra04]. A comprehensive and detailed specification of the HSM, including a comparison with the HIS SHE module, can be found in [WWZ+10] and [WG11]. While the SHE module is comparable to EVITA's light module in its function set, it does not feature the use-flags that are an important building block for our key management approach. The SHE module is envisaged for deployment in cars soon.

4.1.2 Dynamic Key Exchanges

Globally shared keys that would, if revealed, compromise the overall system security are avoided in our solution. The distribution of keys between communication partners is done through an entity called the Key Master (KM). Every ECU e_i on the vehicle network shares two secret keys with the KM. The first key $k_{i,a}$ is used to $authenticate$ the entities against each other, while the second key $k_{i,t}$ is used for $transport\ encryption$ of generated keys. A KM is a logical entity, which can be present on one or more ECUs. It is possible to have multiple KMs (e.g., one for every domain), so that keys are only shared for a limited number of hosts. For explaining the key distribution protocol, we simplified the deployment with only one central KM present in the vehicle network (see Fig. 4.1). For multiple KMs, an additional connection between the involved key masters must

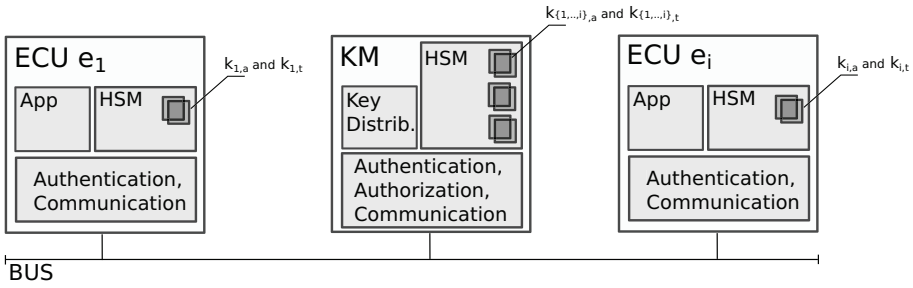


Figure 4.1: One ECU acts as Key Master node (KM). Symmetric keys are shared with every other ECU (e_1, e_i shown). All keys reside in the corresponding HSM (e.g., used for Authentication). System wide policies are evaluated at the key master (Authorization).

be established with the key distribution protocol described in Section 4.1.5. In this setting, a distributed policy system for managing the authorization of such domain-spanning connections is used [SIR+10].

4.1.3 Multi-Criteria Setting of Secure Communication

Our work is integrated into the security framework described in 3.4.1, which notably provides API abstractions for (i) entity authentication, (ii) secure communication (integrity, confidentiality), and (iii) access control along with policy management. The application programmer can open a communication channel from A to B with a given *security-property-set* SP and *security-level* SL. The receiver B may be a group of n recipient nodes, thus creating a $1:n$ communication channel. The security property set contains channel attributes, of which we now use `authentic` and `confidential`, while the security-level refers to the selection of a particular MAC strength, as explained in Section 4.2.3. The communication module initiates the key exchange and establishes a secure channel according to those attributes. Direct and group communication are managed in the access control framework, which interacts with the communication API as well as with the key distribution protocol. In Figure 4.1 the architecture of Key Management is illustrated.

4.1.4 The Protocol

In the following, we develop the protocol for secure communication that is initiated via the Key Master node. We use the protocol in order to distribute a session key for authentication, i.e., the key will be used for generating MACs for further communication. The protocol also applies to the distribution of session keys for encrypting communication, for which the use-flags need to be set accordingly.

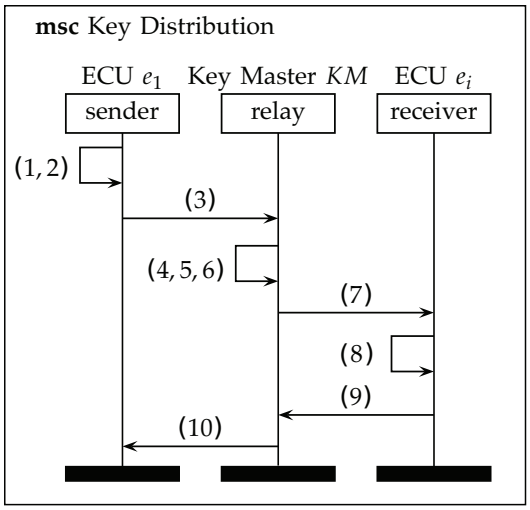


Figure 4.2: Key Distribution via Key Master. Steps (6–9) are executed for all i ECUs in the receiver group. The numbers correspond to the steps defined in 4.1.4.

In order to explain the protocol, a simplified scenario of multicast communication is assumed: one node sends messages to a group of receivers, as it is commonly done in today's vehicles[1].

A secure communication session s between an ECU e_1 and a group g_x of i ECUs e_i is established by proceeding with the following steps.

In Figure 4.2 we illustrate the protocol with a sequence diagram. All steps that affect any cryptographic material (i.e., generation, import and export steps) are executed inside the HSM of the corresponding ECU.

1. $At\ e_1$: Local generation of a key pair $k_{s,g}$ and $k_{s,v}$, where only $k_{s,v}$ is exportable. The key has a limited time of validity (verified upon import).
2. Export of $k_{s,v}$, encrypted with transport key $k_{1,t}$ and authenticated with $k_{1,a}$ We call this the key blob b_1.
3. Open an unauthenticated communication channel between e_1 and KM.
4. $At\ KM$: Authorization of communication request from e_1 to group g_x based on policies.
5. Import of authenticated key blob $b_1 (= k_{s,v})$ into HSM of KM.
6. Re-export the imported key $k_{s,v}$ for all ECUs e_i in g_x with the according transport and authentication keys $k_{i,t}$ and $k_{i,a}$ as blobs b_i.
7. Open unauthenticated communication channels from KM to all e_i.

[1] On the CAN bus data are broadcasted on the medium and received according to a device's acceptance filter bit-mask, similar to IP multicast.

8. $At\ every\ e_i$: Import of authenticated key blob b_i into HSM of e_i: Now the session key $k_{s,v}$ (verification-only) is available at ECU e_i.

9. Acknowledgements for successful import sent to KM.

10. $At\ KM$: Acknowledgement of successful distribution to initiator ECU e_1.

Please note that a strict control is enforced through the HSM's use-flags, i.e., they describe for which operation a key can actually be used for by the HSM. This means that, despite the use of symmetric keys, no ECU can impersonate the sender of a group, which is the sole entity possessing the session key with the sign-flag. This is also the rationale for placing key generation and re-keying at the sender's HSM and not at the key master. Otherwise, a compromised key master would be able to impersonate sending ECUs, as it would also possess the generation key.

We anticipate a limited lifetime of session key material. Specifically, a session is envisaged to last for one drive cycle or a maximum of 48 hours, after which it is not valid for use anymore.

The session key has a limited validity. Thus, replay attacks would only re-distribute the currently valid session key. For readability and clarity of the protocol, we have omitted additional timestamps or serial numbers.

In Figure 4.3 you can find a detailed diagram of how the key is distributed to the individual ECU's HSMs in three phases. The session key itself is depicted in blue and exists in two different variants: With and without red border, which marks the 'sign' capability, i.e. $k_{s,g}$. The blue key without red border may only validate received data, i.e., it corresponds to $k_{s,v}$. The first phase corresponds to steps 1–5 in the sequence diagram. Phase 2 corresponds to steps 6–8, and Phase 3 shows the successful distribution after steps 9 and 10.

4.1.5 Multi-Domain deployment

Central elements of a network architecture often represent so-called Single Points of Failure (SPOF). While it is true that the Central Gateway (CGW), which is part of most modern vehicles since the mid 2000s, already is such a potential SPOF, we would like to emphasize the importance of the availability of security components. If such a security component fails, it may affect the complete system. A fallback to unsecured communication, in our case, would open a door for DoS attacks against the central element. Algorithms for keeping replicate information up-to-date have been developed in Distributed Systems. However an assessment in the automotive domain is beyond the scope of this thesis.

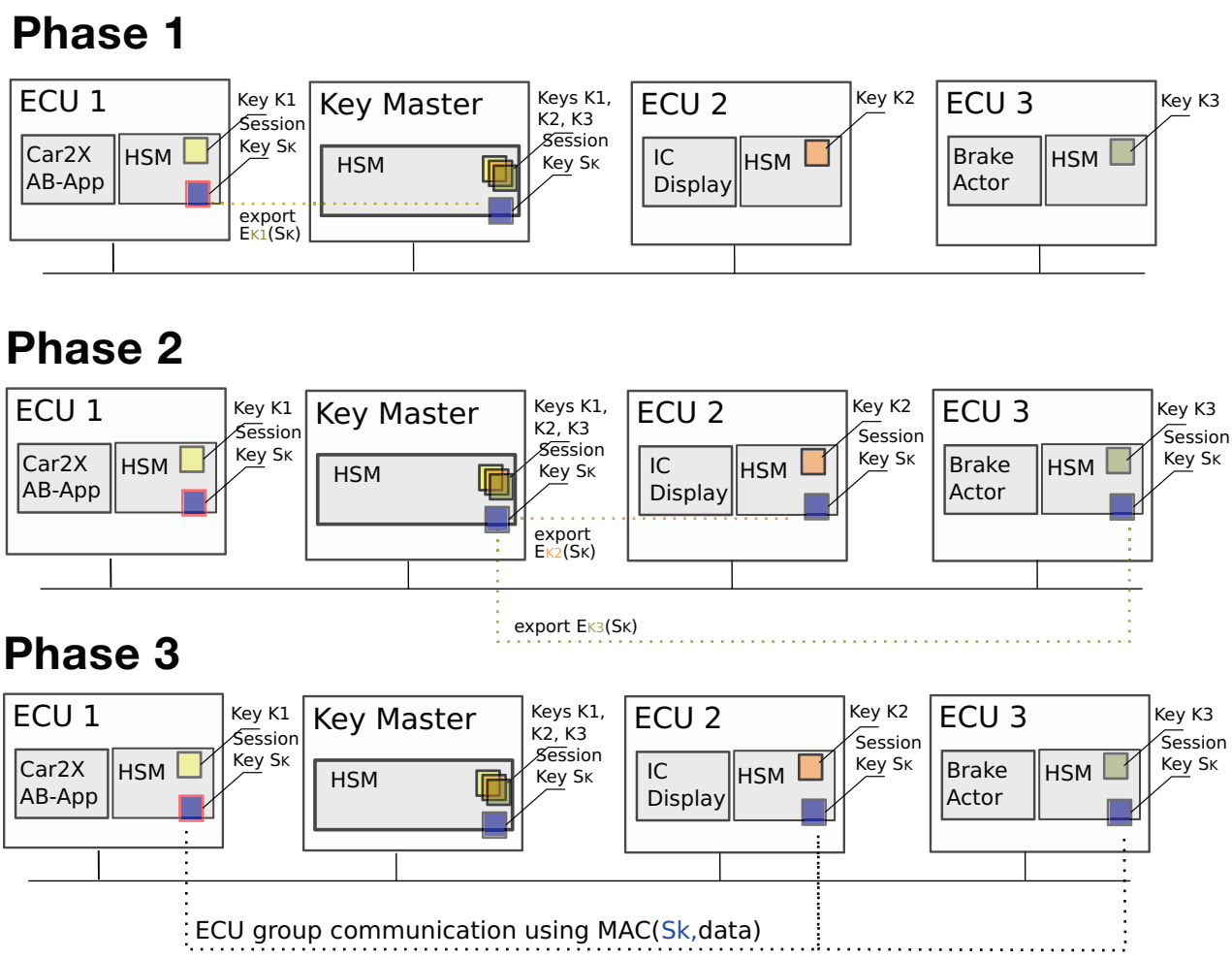


Figure 4.3: Key Distribution Phases: A session key (blue) is generated at the initiating ECU_1 (phase 1). A verify-only session key (without red border) is exported to the master node, where it is distributed to all group members, $ECU_{\{2,3\}}$ (phase 2). After successful key distribution (phase 3), data sent from ECU_1 can be authenticated at all group members. The ECUs as shown were used for the receiving vehicle in the Active Brake use case (Scenario I).

Data Replication

The Key Master node and its HSM may be replicated at another ECU of the network. This would allow for a fallback solution if the primary KM is not available due to failure or attack. The HSM of the secondary KM must possess the exactly same shared secret with each ECU of the on-board network. While the security is slightly decreased due to the fact that keys are stored at a second place (inside the secure storage of the second HSM), the overall availability of the communication architecture is increased. As reliability of vehicular networks is of a great concern, this solution should be carefully assessed. A detailed investigation of this reliability problem is, however, beyond the scope of this thesis.

Partitioned Deployment for Enhancing Domain Security

The approach of using more than one KM node may not only be employed to increase availability through a redundant deployment. It may as well serve the purpose of partitioning the key space for increased security. The vehicle network consists of different domains: The drivetrain domain comprises of engine and transmission control, while infotainment connects display and control elements. This intrinsic design can be used to partition the shared secrets at individual domain controllers: The KM for one domain only contains the shared secret of ECUs of its own domain.

The security of individual domains is increased by this deployment variant. It does not imply that communication must stay inside the domain. We can slightly modify the key distribution protocol in order to establish domain-spanning communication as explained below. This is important for a real world deployment, as current communication groups can also span multiple domains (e.g., the current speed signal is transmitted into nearly all networks).

Domain Spanning Key-Distribution

A requirement for KM nodes is that they are equipped with at least a medium HSM (c.f. Table 3.2 on page 54), so that asymmetric algorithms can be performed on-chip. This allows for a secure channel establishment between KM nodes, even without previously distributed keys (i.e., using manufacturer-signed certificates only).

The distribution protocol for establishing secure communication among different domains of the vehicle only needs to be changed slightly. An additional phase needs to be performed between the KM of the originating domain and the second KM in the destination domain. This additional phase is done after the

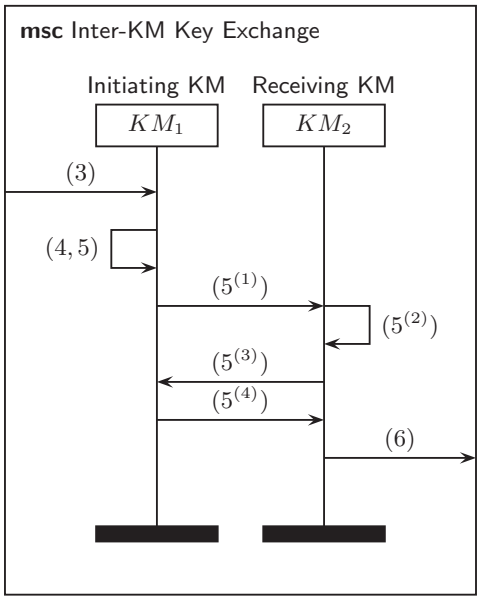


Figure 4.4: Key Distribution in Partitioned Domain: Between steps 5 and 6 of the MSC shown in Fig. 4.2 four additional steps are added. The two KMs of the domains exchange their public key certificates (steps $5^{(1)}$ to $5^{(3)}$) before the actual session key is transmitted (step $5^{(4)}$).

import of the session key (step 5 in the original sequence diagram), and before it is re-exported for the other ECUs of the group (step 6).

The establishment of a connection between two KMs of different domains is based on the asymmetric device key, for which a manufacturer-signed certificate exists. This certificate contains a signature of the manufacturer and is thus accepted at the other KM when used to authenticate, as the manufacturer is a trusted certificate authority. The manufacturer's public key is built into any medium HSM and in order to verify the signature of other certificates. The session key itself is transport-encrypted using the device's key (or a derived key after successful authentication). The process is similar to that of an SSL handshake, without having to agree on a cipher or a separate key, as only transport encryption for the session key in transit is used.

The additional phase is depicted in Figure 4.4. The two domain KMs exchange their public keys in steps $5^{(1)}$ and $5^{(3)}$. In $5^{(2)}$ the receiving KM authorizes the group communication request to one of its domain ECUs. The actual session key is transferred in the last step, $5^{(4)}$, before the protocol continues as originally described in Section 4.1.4. The establishment is similar to that of SSL with client and server-side certificates (mutual authentication), but without the need to agree on a specific protocol or intermediate session key. We have omitted some cryptographic details (e.g., the generation of random number on both

sides) for better readability.

4.1.6 Initial Key Distribution

As symmetric keys are used in the approach in order to keep cost and chip complexity low for the HSM, special attention has to be given to the transport of key material. The initial keys that need to be known at the individual HSM and the Key Master must be initialized in a trusted environment. This means that no external attacker should be able to eavesdrop on communications on the on-board network.

The initialization of the shared keys will take place at the assembly of the vehicle, when also the Vehicle Identification Number (VIN) and other identifiers are programmed into the ECUs. The assembly can be regarded a trusted environment for keying. As a unique key for every ECU is used, the risk of compromise is very moderate, even if an employee on the assembly line would retrieve a key.

4.1.7 Maintenance and Part Replacement

Vehicles need maintenance throughout their lifetime. Although only components experiencing wear and tear need to be exchanged, also electronic components such as sensors or ECUs may experience failure. Exchanging an ECU at a workshop needs special care in our case, as sensitive cryptographic material needs to be replaced or reprogrammed.

In particular, we will look at the exchange of a sensor that must be *paired* with the KM node, i.e., the shared key that comes with a new component needs to be imported into the HSM of an existing Key Master node.

Conditions In contrast to the manufacturer's assembly halls, a workshop cannot be regarded a trusted environment. In addition, the cryptographic material needs to be transferred to the workshop, yet the mechanic should not be able to recover the (unencrypted) key. A further requirement is that network connectivity should not be a prerequisite, as maintenance also needs to be practicable at remote places and in countries with limited network infrastructure.

The approach that is presented here is divided into two scenarios, for online and offline use. In complement to replacing complete ECUs an approach for online updates of vehicle firmware (FOTA, Firmware Over The Air) is discussed in our paper [ISR+11].

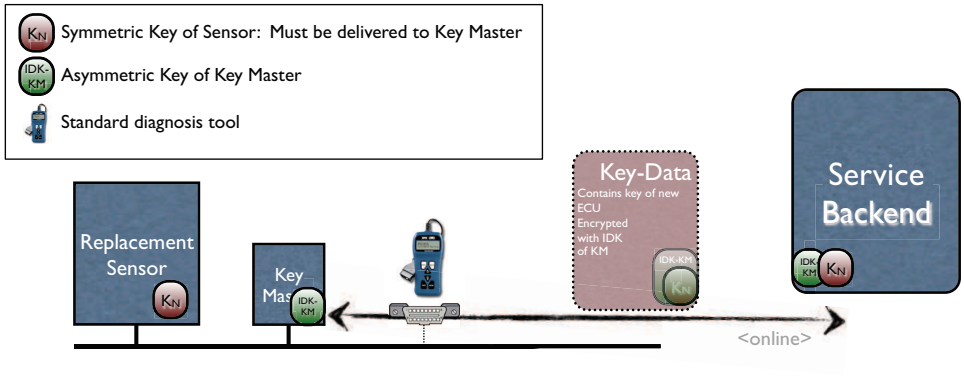


Figure 4.5: Key Pairing for Exchange Parts: Online Case.

Online Case

The dealership or workshop is connected to the manufacturer's backend by means of some secure network connectivity (e.g., a standard VPN or SSL connection), as it is already the case in many countries. This way, the symmetric key of the new component, which is also present in the manufacturer's backend service, can be re-encrypted for the KM of a specific vehicle. A secure handshake process similar to SSL with a client-certificate ensures mutual authenticity to both parties. This is possible, because certificates that are signed by a trusted authority are available at the KM and the backend. The setup is shown in Figure 4.5.

Offline Case

In contrast to the online case, the key information supplied for the new part can not be encrypted for a specific car on-the-fly. An encrypted key is supplied with the product on some media (e.g., CD or flash drive). This transport encryption can be used with specific diagnosis utilities that are equipped with an HSM. An HSM can potentially be implemented in the form of a removable smartcard. This diagnosis equipment can establish a secure and mutually authenticated channel to the KM node of the vehicle and receives its identity certificate (IDK). It then reads the encrypted key that came with the replacement part, imports it and re-exports it for the KM, where it is stored for further use. The setup is shown in Figure 4.6.

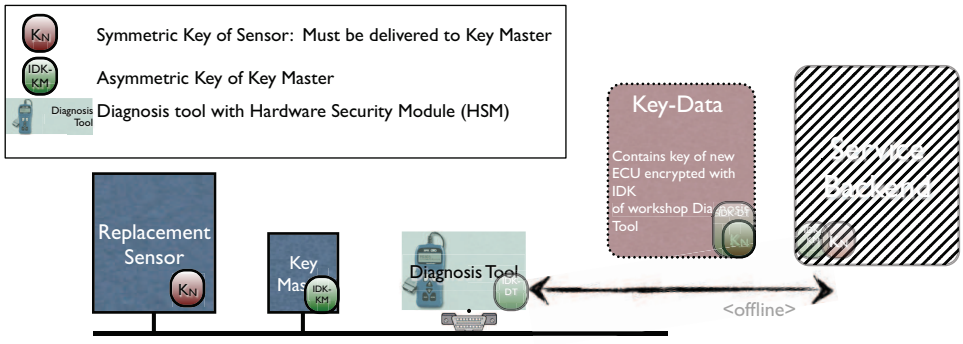


Figure 4.6: Key Pairing for Exchange Parts: Offline Case.

4.2 Securing CAN Bus Communication

The most commonly used communication bus in today's vehicles is the CAN bus that is operated at 500 kbit/s and offers 8 bytes of data payload per packet. From a security perspective, a slice of eight bytes for the security payload does not provide an adequate protection. On the other hand, including a larger payload field or a dedicated field for security is not feasible, as the protocol could not be changed without modification of all CAN transceivers on a bus. There exists a backwards-compatible approach to incorporate security payload by raising the sampling rate, which was validated in an FPGA implementation [HSV11]. However, it is not feasible to change the design of all CAN transceivers in the near future, also given the fact that alternative buses already exist. In the following, we show how we employ a protocol that was primarily used for vehicle diagnosis, the CAN Transport Protocol [ISO04], in order to transmit larger payloads on existing CAN buses.

4.2.1 Technical Background

Controller Area Network (CAN)

The CAN bus is a field bus that has been introduced in 1986. The currently used version is CAN 2.0, released in 1991 [Rob91]. It is known to be very resistant to electromagnetic interference and thus found a wide adaption in vehicular on-board networks. Despite its comparatively low bandwidth (up to 1MBit/s, typically 500kbit/s and 100kbit/s or 125kbit/s depending on the total bus length, are used) it has two interesting properties: message-based addressing and its unique media access control scheme.

Addressing Addressing is special, because it does not rely on sender/destination pairs. Instead, a unique identifier (11bit for standard, 29bit for extended CAN frames) is used. This identifier is used for three purposes:

- To identify the payload,
- To identify the sender (i.e., multiple sending nodes may not use the same identifier),
- To prioritize physical bus access, i.e, media access control on OSI layer 2.

Bus access The bus access on CAN buses is priority controlled: At the beginning of every data transmission, a so-called arbitration phase decides, which station, among those who want to send, may access the bus. Due to the fact

that this arbitration phase is collision free, the media-access-control method is called CSMA/CA[2] for "Carrier Sense Multiple Access / Collission Avoidance". The message priority, i.e., the CAN identifier, decides about the bus access: The lower the identifier, the higher the priority. In fact, the bus itself implements a 'wired AND', where two nodes, accessing the bus at the same time with identifier bits 0 and 1, will sense a 0 due to the 'wired - AND - property' ($0 \wedge 1 \equiv 0$ for synchronous transmission of 0 and 1 of different transceivers of the same bus), i.e., a synchronous transmission of 0 and 1 on the same bus will lead to a 0 being transmitted. At this point in time, the node which wanted to send the 1 will stop arbitration and back off for retrying arbitration at the next frame transmission cycle.

Security Weaknesses One of the main shortcomings of the CAN bus, from a security perspective, is that any participant in bus communication may send any kind of (unauthenticated) data at any priority and thereby hinder normal network traffic. There exists no sender/destination, but a signal paradigm in the automotive world. Data frames are addressed by service, i.e., a specific ID is used to address the recipients of data packets and no sender identification is used. The CAN bus was designed to be (fail-) safe against various electromagnetic influences, but never designed to be secure.

4.2.2 CAN Transport Protocol

We show how a transport protocol [ISO04], which has been used for other purposes in the vehicle, in particular for diagnosis, can be employed in order to transmit additional security payload. This transport protocol segments the payload into chunks of 6 (for the first frame) or 7 bytes (for all consecutive frames) and adds control information, e.g., a four-bit sequence number that is part of the first reserved byte. Thus, a payload of n bytes is split into

$$m = 1 + \lceil \frac{n-6}{7} \rceil$$

packets (for $n > 6$). Figure 4.7 illustrates the segmentation of the payload into multiple CAN frames.

The latency of the overall transmission increases more than linearly with the number of packets sent, as CAN is a priority-based bus in which higher priority frames will always take precedence over an ongoing segmented transmission, i.e., each of the m individual frames can be delayed by higher-priority traffic. The probability of bit-flips increases with the number of segments. According

[2] In fact, the CSMA postfix has never been standardized. Due to that, there also exist CSMA/AMP for "Arbitration on Message Priority" and CSMA/BA for "Bitwise Arbitration" and CSMA/CR for "Collision Resolution".

to the CAN standard [Rob91], the probability of successful transmission of one frame is at least $(1 - 4.7 * 10^{-11})$. As this probability is very high, the loss probability of a message of n segments is low. For small $n \in [2; 20]$ it decreases approximately linear. The transport protocol, as it is standardized, does not know of retransmissions.

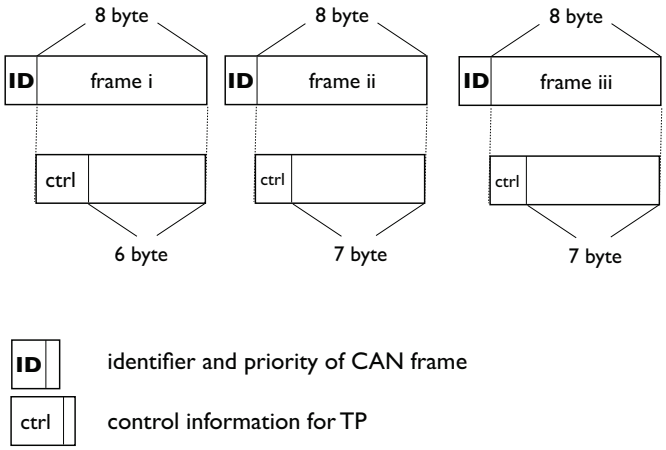
4.2.3 Truncation of Cryptographic Authentication Codes

Not all communications within a vehicle demand the same level of security. Integrity and authenticity are the most demanded requirements for in-vehicle communication. Figure 4.8 outlines an accumulation of low- and medium risk levels, which means that many application security requirements can already be covered with basic security measures, such as truncated MACs. This applies as well to our selected three scenarios: In the first scenario, Active Brake, malfunction or malicious behavior would indeed have potentially fatal effects (leading to strong security requirements). The other two scenarios, playing music and automatic seat adjustment, do not directly have an impact on the vehicle behavior and thus do not have as strong security requirements.

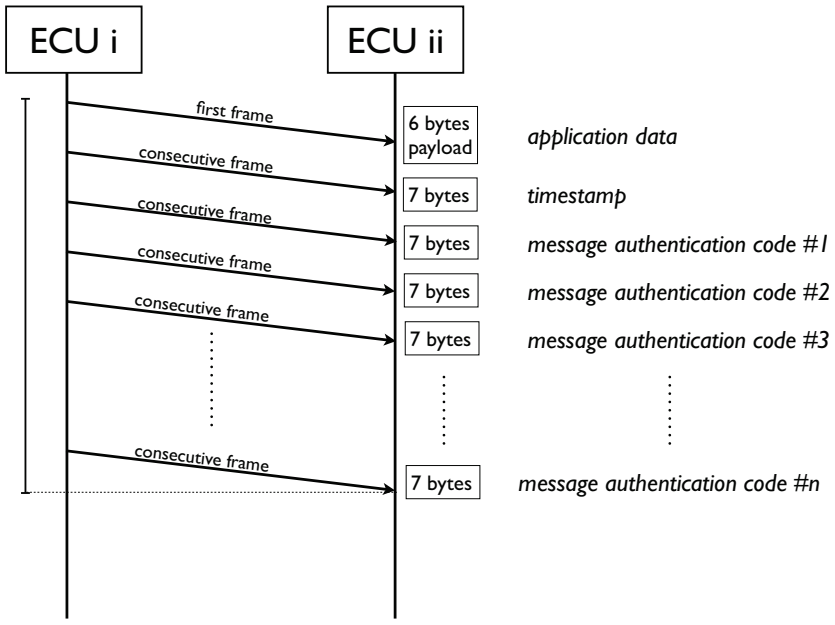
Given the fact that basic security can already protect most applications and that bandwidth is a scarce resource on automotive buses, we allow MAC truncation, i.e., the use of only fractions of a calculated MAC. According to NIST and FIPS recommendations [Dwo05] [Fed02], cryptographic authentication codes should have a minimal length of 64 bits, when no additional measures to limit the validation rate are taken. Within our environment, we already have several environmental limitations, such as bus speed (low) and bus load (high), which hinder attacks at higher speeds. We furthermore set a hard limit for verification trials for any given key-handle to one hundred per second at the HSM. This is the reason, why we can limit the size of authentication codes further and allow a minimum length of 32 bits.

Given these limitations, the expectancy time of a brute force collision attack (i.e., random guesses) for 32-bit MACs at 100 possible tries per second corresponds to over 35 weeks. This is a reasonable value, especially since additional measures, such as plausibility checks involving multiple information sources are usually employed. In Table 4.1 it is shown how different MAC lengths result in increased expectation for success of a brute force attack given the limit of 100 verifications per second.

A requirement for truncating MACs is that the attacker cannot control the way in which the recipient validates the MAC, as discussed in [Mit03]. Our session keys are bound to a specific truncation by the security-property-set that is defined per communication channel. That is, the attacker cannot influence the way the recipient will verify the MAC by changing the payload.



(a) Payload segmentation for the ISO Transport Protocol on CAN frames as defined in [ISO04].



(b) Latency of segmented payload for ISO-TP: The MAC is segmented into n pieces. Every packet is subject to CAN arbitration.

Figure 4.7: The CAN Transport Protocol: Payload, e.g. large cryptographic data such as MACs are segmented into multiple individual frames on the CAN bus.

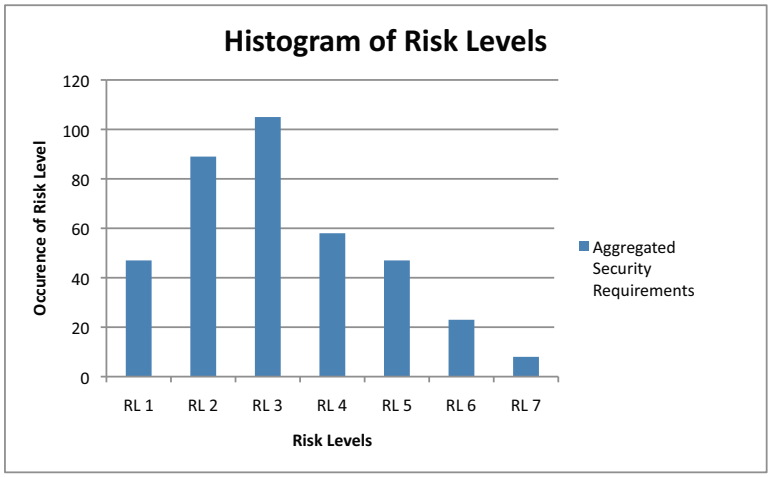


Figure 4.8: Count (bins) of Risk Levels RL1—RL7 with regard to selected security requirements in communication. A risk level is a combination of occurrence-probability and impact. The count shows an aggregate of security requirements for all use cases of the EVITA project (details in [KFL+09, RWW+09, WWZ+10]).

If an attacker would have the possibility to access the API directly, and could choose the truncation length for verification, guessing a MAC would of course be much simpler for reduced lengths. The underlying algorithm that is used to construct the MAC output needs to be crafted in a way that it distributes its output uniformly over the complete MAC-space $\{0,1\}^n$. Usually, this is the case for hash functions and encryption ciphers that are used for this purpose, but there exist some exceptions which must be avoided, or at least carefully assessed, as pointed out in [WFLW08], where they introduce the notion of robustly unforgeable MACs for truncations of the output. A standard AES-based CMAC implementation is used in our HSM implementation.

4.2.4 Implications for CAN bus communication

Clearly, the transport protocol has a strong influence on the performance of communication (i.e., latency and bus load). In order to secure the payload, and also to initially distribute cryptographic keys, the payload is segmented into multiple frames. This means, on the one hand, that individual messages have a higher latency and, on the other hand, that the overall bus load is increased.

We have evaluated the resulting latency and load with two different simulation approaches: TTool and TrueTime on Matlab/Simulink. Our simulations are presented in the Evaluation Chapter in section 6.2, Protocol Measurements. The results of both simulations show an acceptable performance.

bits	expected time to collide
0	n/a
16	5.5 min
24	23.3 h
32	35.5 weeks
48	44,750 years
64	2,932,747,010 years
96	$1.25961 \cdot 10^{19}$ years
128	$5.40996 \cdot 10^{28}$ years
192	$9.97962 \cdot 10^{47}$ years
256	$1.84092 \cdot 10^{67}$ years

Table 4.1: Time to Collide MAC (P=0.5). Random guesses (brute force attacks) for different lengths of MAC at rate of 100 tries per second (intentionally limited by design—restriction applies per key-handle). The expected number of MAC verifications is 2^{n-1} for an n-bit truncation.

Other Buses

The transport protocol can also be applied to other automotive buses. The FlexRay and the MOST bus are newer but also more expensive networks, which offer larger payload fields that do not necessitate any segmentation of a security payload. In contrast, the LIN bus (Local Interconnect Network) is a low cost Master/Slave bus that is largely used to connect low-cost components, e.g., seat-position-servos and control panels with few buttons. Similar to CAN, it only offers eight bytes of payload. The ISO Transport Protocol exists also for LIN to enable diagnostic features. This makes payload segmentation available for all components that support diagnosis. On the one hand, such small components are rarely safety critical and do not need a high security level. On the other hand, the devices that act as gateways are themselves part of an uplink bus (usually CAN) and need to be secured adequately. In particular, for exposed components such as rear-view mirrors or control buttons inside doors, data originating from the LIN network needs to be handled with care, for instance input validation or plausibility checks should be performed.

4.3 Related Work

The vehicle domain has for a long time been driven by functional requirements. Security features have only been added to selected functions, such as immobilizers or remote door unlocking. Recent analyses have shown that the risk of attacks on vehicle bus systems is not anymore of theoretical nature [KCR+10]. Even supposedly well-secured functions, such as remotely unlocking the vehicle,

have been proven faulty [FDC11].

With the integration of new wireless access technologies in order to enable applications based on Car2X and internet communication, new security threats are posed against vehicular on-board networks. Furthermore, in order to enable certain communication-based safety functions, the trust in data from a source has to be assured with dedicated security measures within the vehicular on-board network. While most of research has focussed on securing external communication, the security of the on-board network has been only partly addressed, yet.

For upcoming Car2X communication, several approaches have been discussed in research. Most rely on asymmetric cryptography [KPB+08], but also group-cryptography [GBW07] and timed-symmetric [HL06] mechanisms are taken into account.

For the in-vehicle architecture, numerous authors mention the need for security [Kun08, EB06, GFL+07]. However, rather few real-world solutions have been proposed. A basis for security within the on-board network is the establishment of trust relations by pre-deploying keys, such as the in-vehicle key distribution mechanism which has been proposed in [OYN+08]. Their approach is based on a so-called Key Predistribution System. They use an $n \times n$ generation matrix, which is stored at a dedicated master ECU and parts (the rows) at all n ECUs. As ours, their system uses symmetric cryptography and trusted hardware. However, we see that the practicability of their approach is largely limited by the fact that the configuration must be static over the complete vehicle lifetime. Despite the fact that vehicle upgrades aren't possible without changing the complete keying matrix, maintenance work such as the replacement of ECUs is very complex. Another approach for in-vehicle security is discussed in [Be09]. The presented security framework also relies on a master/slave partitioning. The master node holds a shared symmetric secret with each of the ECUs, similar to our approach. However, only mutual communication is foreseen in [Be09]. These mutual session keys are generated and stored at the master node and do not have the notion of use-flags.

Both approaches are fitted to embedded systems, but pay not attention to an important aspect in vehicle networks. Neither supports communication to multiple receivers, which is the core use case within current on-board networks, where data is sent to multiple receivers. In [GR09], this problem is addressed by creating application groups with dedicated shared keys. This approach also relies on a master node, the Key Distribution Center. A fundamental difference is that asymmetric cryptography is used to distribute the symmetric group keys, and therefore needs more expensive hardware. The approach also uses a master node that possesses the symmetric keys and could thus interfere with communication if compromised.

In order to secure the actual communication [CRH05] proposed the use of RC4

in order to encrypt the bus payload. The frequent renewal of key material is a specific requirement for using RC4, as parts of the key may be revealed if known data fragments are encrypted more than once, as it was seen for weaknesses in WEP. In [NLJ08] it was suggested to compute a MAC over four consecutive CAN frames, truncate the result to 64 bits and split it among the CRC fields of the next four frames. The approach introduces the notion of delayed authentication. A major hindrance for real world deployment is that all transceiver chips of the network would need to be replaced, in order to buffer the appropriate message and to evaluate the CRC field accordingly. Changing the transceiver chips is also necessary for the solution presented in [HSV11]. It incorporates MACs into CAN frames by using a higher physical sampling frequency on the bus. They thereby create extra space for payload, which is used for the MAC.

Our approach has the advantage that it covers both important topics, namely key distribution and transport authentication. We introduce an additional delay in the communication, as other approaches do, but in contrast to others, we do not change the underlying CAN network. Thus, our approach can be applied to standard components that are found in the vehicle today. Our key distribution strategy takes group communication into account, an aspect which had often been overlooked. We use symmetric keys in an asymmetric manner that is enforced through the use of HSMs. Therefore, the required hardware is less complex and cheaper, which is an important aspect in vehicle engineering.

The main publication supporting this Chapter is [SRW+11].

Chapter 5

Dynamic Platform Security

> "You can't defend. You can't prevent. The only thing you can do is detect and respond."
>
> Bruce Schneier

In this Chapter we describe the approach we have taken to enhance the platform security of automotive control units beyond secure communication. It differs considerably from static approaches. While static approaches, such as a secure boot process, or software signatures provide an important building block in system security, they cannot guarantee resistance against the runtime exploitation of firmware or against the compromise of any program running on a statically secured device. We have explained the techniques of these building blocks in Chapter 2.

In the past, we have seen a trend that static security does not prevent the realization of successful attacks for a long time, if the target is of interest for a hacker community. A good and recent example of this trend is shown by Apple's iOS devices, which immediately caught the community's attention when released. The devices are well protected through a sophisticated secure boot process, jailed processes, and cryptographic application signatures as explained in [App12]. Despite this, every version has been successfully compromised for each of the products, sometimes only days after the release. We have shown a timeline of this cat-and-mouse game in Figure 2.10 (page 34).

Similarly, most of the real-world vehicle hacks that were demonstrated in [CMK+11], [KCR+10], and [RMM+10] did not rely on breaking static security measures but on exploiting the runtime system, e.g., by overflowing variables in the music player or otherwise breaking weak implementations, like those for bluetooth pairing and CDMA radio modulation. They also showed attacks against classic

measures by cracking the two-byte access code for firmware flashing. This allowed them to reprogram a modified firmware to an ECU. In summary, these examples of attacks suggest protection means beyond communication security and purely static approaches are required.

For this sort off attacks, the *dynamic platform security* needs to be strengthened. The contribution of this Chapter is an enhancement of the overall system security on multiple levels. We monitor the behavior of individual components in different granularities: With classic Intrusion Detection System (IDS) the behavior of network nodes (e.g., the frequency of messages, message content, or valid addressing) is inspected. This is referred to as *Network Based IDS* (NIDS). Furthermore the behavior of programs during execution (e.g., memory and CPU utilization patterns) is monitored. This is referred to as *Host Based IDS* (HIDS). As the vehicle is a very distributed system of systems, the individual IDS components need to be coordinated in a distributed manner, thus being a *Distributed IDS*.

In addition to the classic Intrusion Detection System approach, tracing data usage dynamically is a promising approach in order to overcome runtime exploitation, especially for potentially untrusted code. Such untrusted code may be third party applications (Apps), where no malicious intent is visible at first sight. This data flow tracking concept is employed in our architecture in order to trace program execution at a fine granularity that goes beyond monitoring CPU or memory usage patterns. We inspect the use of individual pieces of data and follow their propagation throughout code execution. A binary-instrumentation based taint engine is used to locally track data flows in untrusted code.

A generic architecture for the on-board intrusion detection system that is part of our approach is described in the first Section (5.1). The secure communication middleware, developed in the previous Chapter, is used to securely propagate information between the individual nodes of the system. A combination with data flow tracking leverages the classic IDS. It follows tainted data locally, but also among ECUs in order to monitor the information flows between local applications and for distributed applications. We describe this approach in the second Section (5.2). It is particularly suited for ECUs with good processing power, as information flow tracking incurs significant overhead. A new HSM-based approach is described in Section 5.3. It ties dynamic runtime security to the Hardware Security Module, i.e., at a very low level, which is independent from the actual program and operating system execution. It relies on strong time constraints at execution and is therefore particularly suited for small ECUs with statically scheduled jobs.

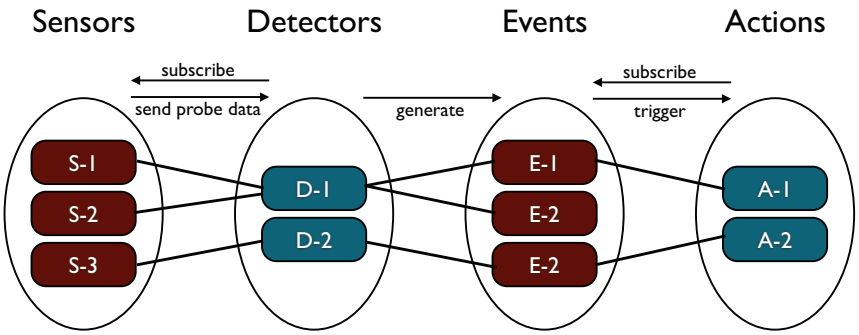


Figure 5.1: The IDS consists of sensors, detectors, events, and actions. Sensors are the source of detection: They send probe data to one or more detectors. Detectors are subscribed to one or more sensors and generate events based on the evaluation of probe data. Events trigger one or more actions, which previously registered for these events.

5.1 Intrusion Detection and Response

Classic intrusion detection systems are based on different types of detection mechanisms and models. They are either specification-based, matching monitored data against signatures or pattern specifications, or anomaly-based, using machine-learning techniques to classify normal behavior and detect anomalies in the behavior of the system by monitoring communication and execution. Similarly, misuse detection models define dedicated misuse of system functions (e.g., a dedicated number of failed login attempts). According to [Bis04], usually a combination of two or three models is used in practice. All techniques can be applied at different layers of processing and communication. Host-based systems, such as traditional virus scanners, historically rely on a large signature database. Since some years, there exist approaches to detect malware by analyzing behavioral patterns at runtime [KKB+06, Rie09, ESKK08].

In the vehicle domain, many of these mechanisms are not applicable as-is. Communication patterns can be detected and classified more easily, as electric signals are often repeated in short periods and both the format of network frames and the data format are well-defined. However, common ECUs often do not have the processing power to run sophisticated algorithms, so that simple classification (such as analyzing the periodicity or outlier values of a signal) is preferred.

A fundamental difference between classic intrusion detection and automotive systems is that patterns to detect intrusions cannot easily be updated in vehicles. While today's virus scanners regularly (daily or at most weekly) update their signature and heuristics databases, such a reliable back-end connectivity is not foreseen for all kinds of vehicles, especially in the low-cost sector.

In the scope of the security framework described in Chapter 3, we have created

intrusion sensors that can be distributed to different ECUs of the vehicle network. Their observations are sent to a master node, where they are evaluated. Our approach leaves the flexibility for IDS evaluations to be either pattern-based or behavior-based. The evaluator functions can be stateful (i.e., save a time-window of received probe data) or stateless (i.e., only evaluate the last probe received).

If the evaluator function suggests a possible intrusion or abnormality, a labeled *event* is triggered. The label depends on the potential intrusion that was detected.

At the last stage of the IDS, these events trigger *actions*. An action is subscribed to one or more events, in order to react on specific conditions. The actions can have an active influence on the configuration of the vehicle network, e.g., modify the system-wide policies, or signal the detection to some display or log inside the vehicle. In any case, the design of events and actions must make sure that false positive alerts do not create an additional attack surface for Denial of Service (DoS) attacks, and do not create confusion or distraction for the driver. This is a sensitive topic which we take up the discussion on intrusion response in the conclusion (Chapter 7). Fall-back rules that isolate parts of the vehicle but guarantee a safe and drivable state of the system may provide a viable solution.

The relations between sensors, evaluators, events, and actions are visualized in Figure 5.1.

5.1.1 Architecture: Vehicular Network and Host Based Intrusion Detection System

Deployment The overall architecture of the Intrusion Detection System developed within the security framework is partitioned into *sensors* and *actions*, which are deployed at different places within the vehicle. The actual evaluation that consists of *detectors* and *events*, is done at a central place at the EMVY master node (see 3.4.1 on page 49 for details on the architecture).

As the networks used within the vehicle are broadcast media, network-based anomalies can be detected, but their origin cannot be identified. A deployment of IDS sensors at all ECUs is more expensive, but would be suited to determine the misbehaving node with a voting-like approach, as discussed in [LNJ08], where all nodes of the system were equipped with sensors.

An example layout for sensor deployment is shown in Figure 5.2. In particular interfaces with other buses and with the outside need to be protected. Such interfaces include wireless links, such as Bluetooth and WiFi, but also wired interfaces such as OBD-2 (accessible from inside the vehicle) and even those that are possibly accessible (e.g., cables connecting electronics inside the door or side mirrors). There exist tools to inhibit the alarm trigger and roll down

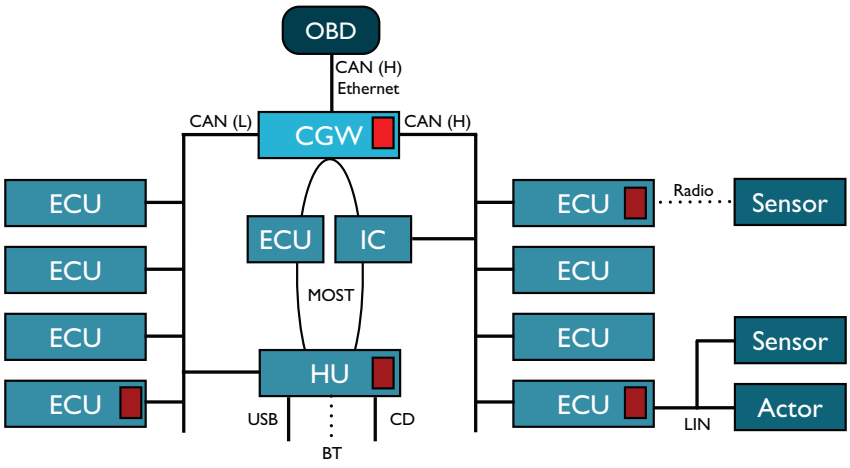


Figure 5.2: Distributed IDS Deployment in Vehicular On-Board Networks. Shown is a simplified network. ECUs are connected by two CAN buses (low and high speed), one MOST bus, and a LIN bus. The Head Unit (HU), the central infotainment component, connects external media devices via USB or Bluetooth. The red boxes show an example deployment of the IDS sensors. IDS sensors should be placed wherever the risk for a potential attack is increased, e.g. where external interfaces are connected. The light-red box constitutes the IDS master, which evaluates probe data and generates events in order to trigger actions. It is placed at the CGW, as the most connected entity of the network.

the windows of a theft system when opening the door electronically, only by hooking up to the wires [Jac12].

IDS Sensors

The *sensors* take measurements of their environment and, in a periodic manner or on occurrence, send these data to the master node. Data may be preprocessed within the nodes, in order to aggregate data over a certain time window, e.g., for one second or one minute.

The probe data that is sent to the detector functions consists of:

- A type field that identifies the kind of probe data from this sensor,
- An issuing entity identifier of the sensor, including address information,
- A timestamp,
- A data field that contains the actual probe data.

Sensors can be host and network based. *Network-based* sensors consider, for example:

- Frame-Format (packet conforming to the expected standard),
- Frame-Content (packet content is not within the expected value interval), and
- ID-Frequency (How often are frames of a specific ID sent).

Host-based sensors that may apply to the whole platform or individual tasks, measure for example:

- Memory-Usage (or usage patterns)
- CPU-Usage (or usage patterns)
- DIFT alerts (as explained in Sect. 5.2)

IDS Detectors and Events

The *detectors* are placed at the master node. They subscribe to one or more sensors, which are distributed over the network. In addition to the probe values from sensors, a detector has further data sources: It uses a configuration (e.g., a detection threshold may be set), and may also takes advantage of a database, such as a set of signatures or patterns, or learned knowledge for clustering-based approaches.

The policy module is a specific datasource. It stores system-wide policy rules, which define which kind of communication or service is permitted to which entities and on which network segment. Based on these rules, an evaluator function can detect if a policy was overridden, e.g., by some kind of network communication.

A detector's evaluation function processes the received probes based on its model: The model consists of configuration and, optionally, a database. If the evaluator function detects anomalies, it generates an *event*. An event is basically a small data structure that consists of a timestamp, the origin, and the event type. In addition, a reference to the original probe data is contained. The original probe data is not discarded, but saved until it is handled by an action in the next step.

IDS Actions

The *actions* are the last item in the chain of the IDS components. An action can be as simple as saving the triggered event and the corresponding probe data. On the other hand, it can be as complex as changing system-wide policies. In any case, changing the vehicle behavior or functionality is a delicate topic, as it may surprise or distract the driver. Actions are subscribed to events. The

actions can be placed either at the master node, or at any other place in the network.

Actions include:

- Logging the event and potential probe data,
- Alerting the driver or some back-end,
- Changing system-wide policies,
- Shutting off services and ECUs

If there is, for any event, no specific action that is subscribed, the *default action* is always to log and save the event.

5.2 Distributed and Dynamic Information Flow

In the vehicle, data are constantly exchanged between different units via on-board networks. While the communication between sensors, controllers, and actors was more or less a closed system, the integration of consumer devices and installable applications creates a more dynamic environment. These new features demand an access to vehicle services, which were historically isolated from any external communication. Furthermore, vehicles will soon receive data from other vehicles or from the infrastructure via Car2X communication. Such data might be crafted maliciously in order to exploit protocol stack vulnerabilities in the vehicle.

We would like to emphasize two types of attacks that can go beyond the capabilities of static security measures: i) hijacking the control flow of an application and ii) obtaining and using data in a way that infringes the driver's privacy.

We exemplify them with the scenarios from Chapter 3. The first type of attack, hijacking the control flow of an application, comprises classic exploiting techniques, *in order to execute different code*. The second type consists of applications that *access potentially delicate data* (such as the seat position in scenario III, or even more so, internal microphones) and, at the same time, have capabilities to record or transmit data (e.g., to retrieve content or updates from the internet). This breaches the user's privacy by secretly transferring private data or information about behavior or location. Such a behavior has been largely observed for applications on mobile devices, e.g., out of 30 popular Android applications 15 would transmit additional personal data (Oct 2010) [EGC+10]. A similar study based on automated static binary analysis of iOS applications revealed that 195 of 1407 applications (14%) would secretly access the device identifier (Jan 2011) [EKKV11].

Enck et al. use an approach for analyzing Android applications by tracking data at runtime [EGC+10] that is also interesting in our context. Their approach tags

data at the source, e.g., system calls, and follows the tagged data throughout the execution of a program. While the approach of tagging data as *tainted* was not new, this approach was the first to show that an implementation on embedded devices is feasible. Tainting offers the advantage of knowing about the origins of data without statically following its course of execution, which is sometimes not easily possible, for example in binary code.

However, this concept cannot be directly adopted in the vehicle domain. It could possibly be used for a stand-alone controller or computer, such as the vehicle's head unit. The conception of vehicle networks is, to a great extent, a connected system of distributed sensors, actuator, and computation nodes. In general, the idea of tracing data through a computation system seems appealing. We use of this concept not only to address privacy issues, but also intrusion prevention or detection.

The system that we propose uses the taint of data also outside of a single execution environment, i.e., it propagates tag information between ECUs within the vehicle network. By this, we can track the origin of data in order to *avoid unwanted leakage of private information* to third parties, and can also *ensure validity of received data* by matching the required route of processing with incoming (ingress) and outgoing (egress) rules and policies.

To illustrate the overall concept, Figure 5.3 shows data generated at the brake controller (red) and GPS receiver (green), used at the ESP/ABS unit and sent along to the Head Unit. A warning message is generated for the Car2X interface, and then to the Instrument Cluster (IC). Data is combined with location information (green) by the Car2X unit before being sent to other vehicles.

A combination with classic access control policies allows to protect against any unwanted leakage of private information. By tainting inbound data sources, a taint tag in combination with a ruleset also becomes a tool to detect intrusions.

Preconditions Two preconditions have to be met in order to rely on such distributed taint data: i) communication security and ii) static platform security need to be assured for all nodes in the processing sequence. These conditions are met by the secure communication protocols developed in Chapter 4 and the EVITA Hardware Security Module described in Section 3.5, which we use as building blocks.

To be able to combine and propagate taint information locally, a dynamic information flow engine is used. It propagates taint tags among instructions during processing. Unlike the approach on a single execution platform, propagating the information to remote, intermediate processing nodes, requires the authenticity and integrity of the taint tag.

In addition to privacy related information leakage threats, information flow control that is analyzed at the scope of single processor instructions can also be

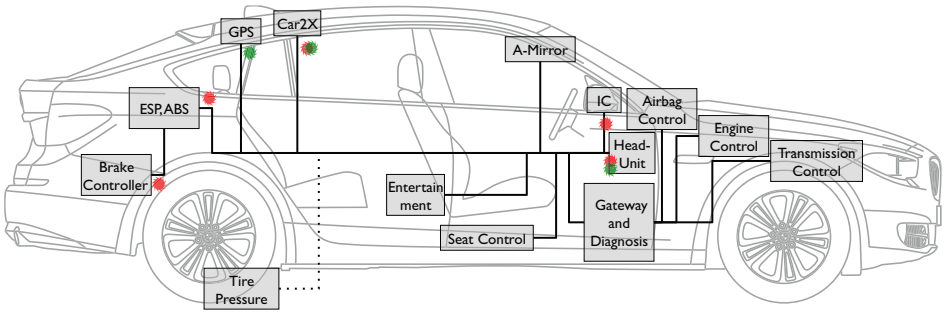


Figure 5.3: Data originating at the brake controller and GPS unit is enhanced by a taint tag (Scenario I, sending vehicle). Background ©BMW.

used in order to protect against classic attacks on the execution architecture, such as buffer- and heap overflow attacks. In such cases, the user-supplied marking of control-flow related data is checked before a call (e.g., return or free) is executed, thereby making it possible to spot the attack.

5.2.1 Data Flow Tracking

On-Board applications are designed in a particularly distributed and communication centric fashion. Applications are built as small function blocks following a component based model that emphasizes the connection of inputs and outputs to basic or composite software modules. Those software modules are finally mapped to different ECUs, thereby generating specific communication patterns. Automotive middleware frameworks such as AUTOSAR are generally used to support such architectures.

In this distributed software and hardware system, it is important to follow data not only within, but also between individual ECUs. For example, a successful attack via the tire-pressure monitoring system as shown in [RMM+10], in which the sensors transmit their data wirelessly and unauthenticated, makes it evident that superficially unrelated data can compromise parts of the vehicle. In the case of their experiments, the speedometer display was compromised. Data like tire pressure sensor readings are also routinely broadcasted to the in-vehicle network and distributed to further ECUs. These data may be used as inputs for plausibility checks within the electronic stability control module or to trigger the "limp home" functionality (in case of a supposedly flat tire) within other units. If the original data has been maliciously crafted, an ECU using these data could be subject to an exploit, whereby an attacker can gain control of the execution by diverting the CPU's control flow. A possible overflow in a device like the electronic stability control can result in the brake system effectively being compromised, as it directly and individually controls brake actuators at all wheels.

Layered Architecture The layered approach of the architecture is shown in Fig. 5.4: Our approach makes use of the information flow on the level of network and inter-process communication as well as internal API calls for accessing an ECU's I/O. This is hedged by the middleware framework, which allows to track in- and outgoing information. Incoming data are tainted with tags we introduce. We verify the presence and absence of tags for outgoing data according to the ruleset (introduced in the following Section). A major part of the system is the taint engine, which makes binary instrumentation: It instruments all applications and libraries at runtime and takes care of the propagation at runtime. Tags are propagated among the ECU's registers and memory at execution time. As different kinds of input-sources need to be tracked, we use distinct taint tags, an approach often called "colored taint" [EKS+10].

5.2.2 Binary Instrumentation for Taint Tracking

In order to introduce taint tags into running applications, we make use of a technique that is called "binary instrumentation". This technique injects custom code to binaries at run-time, i.e., one can instrument machine instructions and system calls in order to follow the flow of data between registers or memory regions, as well as to take precautions if data is used in a questionable manner. One of the major advantages of binary instrumentation is that the source code of analyzed programs does not need to be available, because the processor's instructions are directly instrumented at runtime. This enables monitoring programs of unknown pedigree for compliance with given rules. We make use of Intel's Pin [LCM+05], combined with a taint tracking engine. Pin is a generic tool for dynamic binary instrumentation and provides the base for dynamic taint tag analysis. Pin is available not only for x86 architectures (Windows, Linux, Mac), but also for ARM. We have conducted the experiments on IA32 Linux.

There exist a number of prototype implementations for host-based dynamic data flow tracking based on Pin [CLO07, KPJK12, ZJSK11]. We analyzed these prototype implementations for feasibility to integrate with our security framework. We use the middleware from Chapters 3 and 4 to trace data through execution on the platform and to propagate taint tags together with network streams within the vehicle.

5.2.3 Data Flows: Access Control

Data can be exchanged in a multitude of ways.

- By accessing shared memory,
- Via filesystem access,

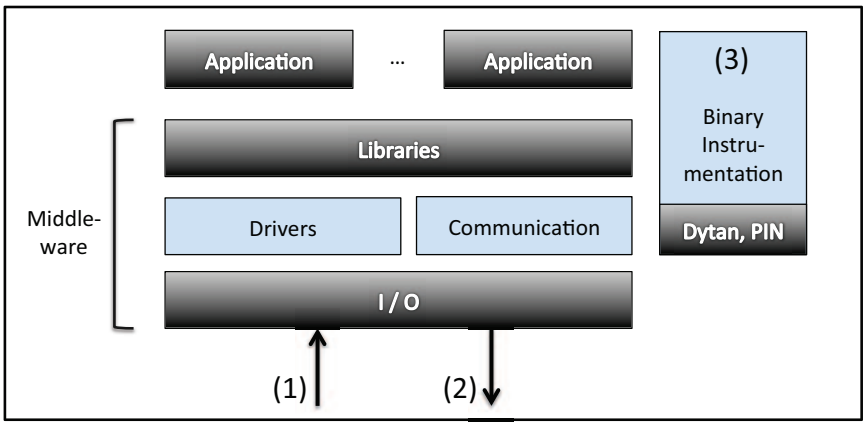


Figure 5.4: Layered architecture. Applications use libraries, which use corresponding drivers and communication functions to exchange data. At the driver and communication layer, $data\ that\ is\ read$ (1) is tagged with an appropriate label, e.g., $sensor_n$. The label is kept in a shadow-memory and propagated throughout execution. Dytan instruments application binaries and libraries at runtime, which propagates tags in memory (3). When $data\ is\ written$ (2), the rules are enforced, i.e., it is analyzed whether data only contain information from allowed sources (i.e., the tags correspond to those expected in the rules).

- Through inter-process communication,
- By using network frames.

We do not take covert channels into account, although this is possible in principle. To limit possible data and information leaks from processing functions, we first have to define which are the subjects in our model, i.e., at which granularity data flows should be described as rules. Data flows exist in various granularities (from fine to coarse): Between instruction arguments, in function and system calls, between local programs of one ECU, and between ECUs on a network. If control flow propagation of taint tags is taken into accounts, it would, on the one hand, cover indirect information flows. On the other hand, by propagating a tag of a variable inside a conditional statement to all variables changed within the conditional code block, a taint tag is easily widely propagated, leading to a state called 'over tainting'. For this reason, an indirect propagation of taint tags is often not considered.

While we cannot limit general processor instructions (e.g., moving between registers and memory), the information flow at instruction level can be monitored and tracked. This allows to follow the information flow with a fine granularity and to enforce control on data on the stack, e.g., to verify that the return address of a function call has not originated from anywhere else than the function call itself. Therefore, a large amount of potential attacks can be eliminated, in particular those based on overwriting the stack pointer, e.g., buffer-overflows [Lev96],

format-string [scu01], and return-oriented-programming (ROP) [SPPG04] exploit techniques. A function-based propagation, if we would not have the ability to instrument and trace individual instructions, would quickly result in over tainting, because all outputs of a function need to be marked according to all input tags of the function. This is due to the fact that the dependencies between inputs and outputs can only be detected with instruction level information flow tracking. In addition it would not be possible to directly prevent exploit techniques on the stack.

5.2.4 Taint Based Security Policy

It is not adequate to build access control rules for individual processor instructions. Likewise, access control at the network level is not adequate, as it is too coarse (see Fig. 5.5 for an example). The adequate granularity to describe access control based on taints is tightly linked with the communication structure of programs and functions: The basic principle of this approach is to control which program may communicate what $kind\ of\ data$ with which partner. This approach is naturally modeled by a graph. In addition to representing functions as vertices v and communication between them as edges e, we also label the edges with the $kind$ of data carried. This allows to distinguish whether a certain data exchange (the edge of a graph) between programs or functions (the vertices) is allowed or not. The labels of edges model the tags allowed for this particular communication, described by e, and consist of a set of taint tags $t_e \in \mathbb{T}$, where the set $\mathbb{T}$ contains all possible taint tags t in a system.

Definition: A $tainted\ graph\ for\ information\ flow\ tracking$ is defined as a set of vertices V, a set of edges E, and for each $e \in E$, a set t_e of tag labels. A graph is thus a triplet $\mathbb{G}_{IFT} = (V, E, \{t_e\}^{|E|})$. Each tag set t_e contains $[0; n]$ tags, with n being the maximum number of tag sources.

A vertex v represents an entity, which communicates over one or more edges. An entity can be a function call (in fine granularity) or an ECU (in course granularity). Data sent through these edges originate from the content of internal variables var of an entity. We define the function $\mathcal{T} : VAR \rightarrow \mathcal{P}(\mathbb{T})$ that assigns to each variable $var \in VAR$ the set of tags t_e belonging to edge e.

The information flow graph can be seen as a ruleset for communication. It determines the allowed tags for each communication channel e.

A rule r consists of the association of an edge together with the allowed tags t_e. Therefore, the a rule is constructed as $r = (e, t_e)$, where $t_e \subseteq \mathbb{T}$.

All vertices at the borders of the graph, i.e., those which have either no incoming or no outgoing edge, serve a special purpose. They are the $source$ or $sink$ of information. A source may be a sensor (i.e., data obtained through I/O) and a sink may be an actuator.

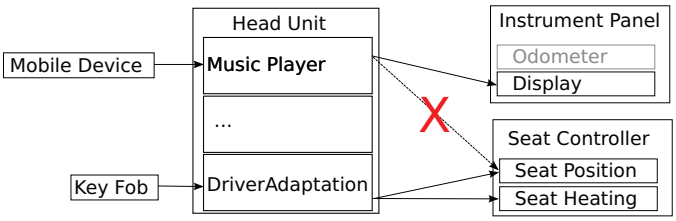


Figure 5.5: Multiple applications are located at the vehicle's head-unit. They have access to different services via the network. Access control must be detailed at the level of applications, as network-based rules would not suffice to detect and prevent malicious messages, e.g., an attempt from a compromised music player to control the seat position. The music player is not malicious per-se, but may be turned against the system by exploiting potential software vulnerabilities with specially crafted input files, as shown by [CMK+11].

Example: Figure 5.6 shows an example of a tag propagation tree for the driver adaptation comfort function. This example consists of four rules: two in order to allow individual data flows from i) an RFID reader that receives the key fob's identifier and ii) a file on disk (both are local), and two rules to allow data streams from the driver adaptation application to two different services for seat controls, both allowed to carry data from corresponding input sources (t_1 and t_2). The latter two rules relate to network data, while the first two are local to the ECU.

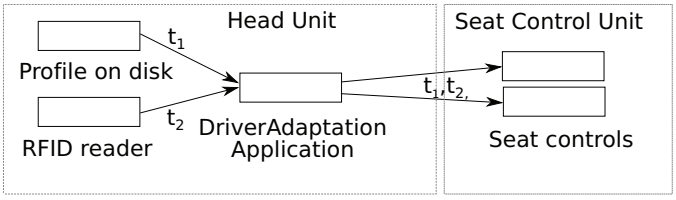


Figure 5.6: Propagation tree for DriverAdaptationApplication. Data is tainted at source nodes (t_1,t_2). This corresponds to an I/O operation at the Head Unit. The application issues commands to the seat control unit via the network. The combined tags are conveyed over network communication and evaluated at the receiving side before the actual operation is triggered. They are also propagated throughout operation, to ensure maliciously crafted data do not change the application control flow.

Baseline rules These rules have to be defined by the manufacturer *a priori*, e.g., as expert-knowledge along with the functions that are deployed. For the majority of vehicle functions, this is unproblematic, as communication channels are known already at the design-phase of the on-board network and usually exist as structured data. For the vehicle architecture, the K-matrix describes communication on the CAN bus. In the upcoming development paradigm associated with the AUTOSAR operating system, individual data exchange between pro-

grams and functions are modeled. One of the concepts within AUTOSAR is an abstract communication paradigm. The VFB models communication between programs and functions, which are later grouped into composite functions and finally mapped onto concrete ECUs. From this deployment mode, all necessary network and local communication are generated into the middleware configuration. This facilitates the generation of the proposed ruleset, as all internal communications of control units are already known and described explicitly.

Application specific rules For third party applications to be loaded onto the head unit, a security ruleset also needs to be supplied along with the application. This ruleset must be evaluated against the system's policies, i.e., the baseline rules. Furthermore the driver may grant certain additional capabilities, which are restricted by the general context that is defined by the manufacturer. This can be compared to Android's security manifest but is more detailed. While the granularity for Android security rules is very coarse (e.g., an application requires network access or not – either full access or no access is granted). Our rules include the kind of data that was used to generate specific I/O operations, such as network requests. The example of Figure 5.6 depicts the control request, which carries tags related to i) the key fob identifier (read via an RFID reader) and ii) the user profile (read from disk). This allows to effectively limit access to the vehicle functionality and leakage of personal data.

5.2.5 Example of Tag Propagation

The propagation of tags happens at every intermediate processing step. For the purpose of controlling the origin of (possibly untrusted) data, we use the unification of tags as a common propagation-logic, as described in the next subsection.

The example given in listing 5.1 is a short application that reads data from two I/O sources, a sensor (possibly memory mapped), and a file. In both cases, the "read" call is used. It then combines the data through the application of a function `f()` and, as a last step, writes data to some output handler, e.g., via the network.

Listing 5.1: Application Code Example for Taint Tag Propagation

```
1 data x,y;
2 data z;

4 x = read(sensor-input);
5 y = read(file-input);

7 z = f (sensor-input, file-input);

9 write(z);
```

At the end of the execution the taints for the memory regions of variables `x,y,z` are: $\mathcal{T}(x) = \{t_1\}$, $\mathcal{T}(y) = \{t_2\}$, and $\mathcal{T}(z) = \{t_1, t_2\}$.

While the program is executing, the taint engine instruments all calls so that the tag is set at the `read()` calls and $propagated$ at the call to function `f()`. Depending on the source of data, the `read` call results in different taints $\{t_1\}$ and $\{t_2\}$ being set. They represent input from sensors and file input in this example.

Memory is tagged byte-wise. This means that not all bytes of `z` need to carry the same taint tag. This depends on the function `f()`: If, for example, `x,y,z` are strings and `f` is a concatenation function, then `z` contains bytes from `x` and also from `y`. This means that the memory for `z` consists of two regions from `x` and `y` that are individually tagged with either $\{t_1\}$ or $\{t_2\}$, for instance if `z` is a structured record. If `x,y,z` are primitive variables, and a calculation depending on both inputs is performed (e.g., $z = x + y$), the variable `z` will be tainted with both tags at the complete memory region ($\mathcal{T}(z_i) = \{t_1, t_2\}$ for all i bytes of `z`).

The taint is aggregated over the complete memory region as described in the next subsection only if data is sent via the network in one piece. This is done primarily for limiting unnecessary network overhead that would be inevitable if every byte would carry its own taint-set.

5.2.6 Network Marshalling

In order to follow data throughout the execution over multiple processing stations, our framework automatically adds the according tags for data contained in network messages and signs the message with a previously exchanged, application-specific symmetric session key. We use a low-cost symmetric message authentication code (MAC) to secure the payload. This MAC, in combination with a key distribution protocol as described in Chapter 4, makes distributed taint tags resistant against man-in-the-middle attacks. Depending on the application's requirements, the protocol adds a sequence number or a timestamp that provide replay protection.

A set of tags is added to the payload of a communication message. In order to limit the message size, one-byte single taints are not sent as-is, but instead combined and aggregated. This aggregated set is taken over all memory areas of all message segments (i.e., individual sub-messages or variables). This means that, for all n segments of a message frame msg:

$$T_{msg} = \bigcup_{i=1}^{n} \mathcal{T}(s_i).$$

is sent over the network. Before being sent, the tags of the data are checked against the corresponding ruleset. The taint information is included in the message header as depicted in Figure 5.7.

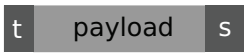


Figure 5.7: Network data is combined with a set of tags and an application-specific cryptographic signature. As communication is always handled by the middleware, the application only receives the (already internally tagged) payload.

5.2.7 Multi-Level Enforcement

The enforcement in our system according to the tainted information flow graph happens at two different layers:
First, an *immediate enforcement* happens at certain instructions (e.g., RET, JMP) and system calls (execve, fork, etc.). Here, the requirement is that no tag is present at the according arguments (e.g., the called address or, in case of RET, the return address). If there was a tag present, this would mean that this piece of data originated from outside the actual execution, which is prohibited in the system to avoid control flow hijacking. This monitoring is directly performed within the binary instrumentation, i.e., the taint propagation engine that has hooked info these instruction calls.

Secondly, *middleware enforcement* is employed whenever functions are called through the middleware. This includes calls to send data over a remote communication channel or to save data to disk. Input data are then analyzed for the *presence* and *absence* of tags corresponding to the according rules. For example in Scenario III, this means that tags $\{t_1, t_2\}$ and no other tag must be present when the `DriverAdaptationApplication` sends requests to the seat control unit (cf. Figure 5.6). In the architecture schematic drawn in Figure 5.4, (3) corresponds to outbound communication: This is where middleware monitoring and enforcement applies.

The taint tags themselves are stored in a separate memory region referred to as *shadow memory*. By instrumenting calls, it has to be assured that no explicit write access to this memory region can be performed by compromised applications. Thus, the integrity of taint tags is ensured.

5.3 Timing Based Hardware Security

In many automotive ECUs, typically the smaller ones, the scheduling of tasks is nearly deterministic. For example, functions read inputs and write outputs in a periodic and timely constrained manner. This strictly static scheduling can be used as an orthogonal dynamic security mechanism. In the following, we propose a solution that makes use of this feature by restricting the usage of keys inside the HSM.

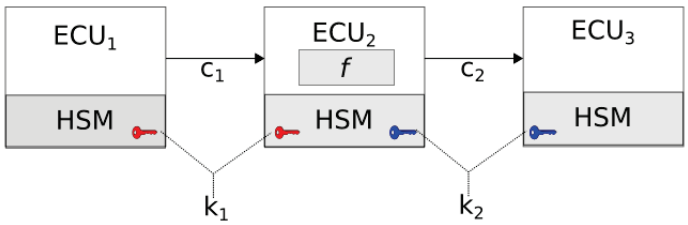


Figure 5.8: Runtime Checks by Controlling Key Usage: The keys k_1 and k_2 are used to secure communication links c_1 and c_2.

5.3.1 Timed Key Usage at Hardware Module

We use the Hardware Security Module as a trusted device to measure the runtime integrity of the platform. In Figure 5.8, we show a common communication pattern between ECUs, which takes place within a given application, e.g., Active Brake. ECU_2 receives a message c_1 from ECU_1 that is secured with key k_1. A function f on ECU_2 subsequently evaluates the received message and sends an output message c_2 to ECU_3, which is secured by k_2. The key k_2 is only used in this context, i.e., a message c_1 has previously been received and processed.

If the HSM on ECU_2 knows the expected processing time $calc$ of the processing function f, which outputs c_2, it may measure the time between verifying c_1 using k_1 and signing c_2 using k_2. Let t_{calc} be the actual time needed for processing f.

If $|calc - t_{calc}| < \delta$ does not hold, then f may have been corrupted.

Figure 5.9 depicts the communication and processing of f at ECU_2 and its HSM.

In general, for every function f that uses this communication pattern, the HSM of an ECU stores the according pair of verification and signing keys k_v and k_s, the exact processing time of f, $calc_f$, and a delta value δ_f.

This constitutes a ruleset $r = (k_v, k_s, calc_f, \delta_f)$ for each function f using this communication pattern.

If a rule is violated, this is reported to the original ECU or, if existent, on a secondary secure channel to a higher level system such as a neighboring IDS node.

5.3.2 Requirements for the HSM and Discussion

In order to enforce these rules that are based on the usage of key material, additional information needs to be maintained inside the HSM.

- A state, i.e., when has this key been used,

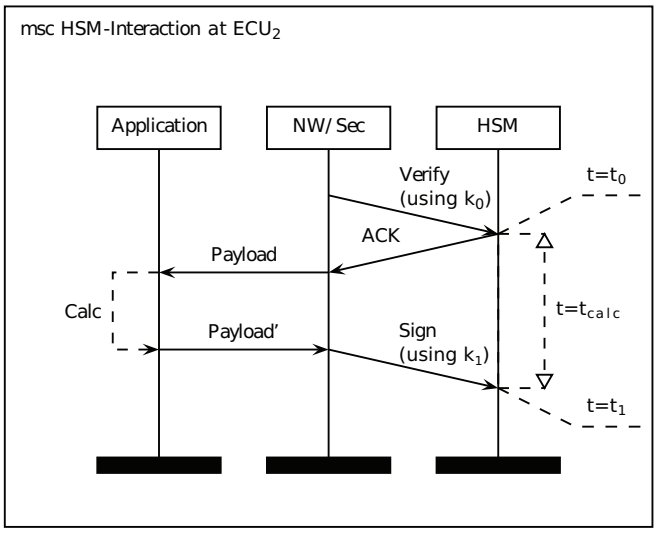


Figure 5.9: Behavior check of key usage at HSM on ECU_2: key k_2 must be used exactly after t_{calc} has elapsed from the previous usage of k_1.

- A timer, i.e., to measure the time elapsed between different commands and key usage,
- A usage rule store.

The steps taken by the HSMs are as follows:

i) SELECT the appropriate rule according to the verification key k_v.
ii) VERIFY the input and take time measurement (t_0).
iii) SIGN the output and take time measurement (t_1).
iv) CHECK the satisfaction of the time constraint for the usage rule $(|calc_f - (t_1 - t_0)| < \delta_f)$.

If the check fails, the signature result may be suppressed and an alarm raised. The HSM module must have a control output to be capable of signaling the violation of rules to the ECU, or possibly even to another external instance, e.g., another ECU.

Discussion

This approach focuses on small ECUs with static scheduling. It cannot be applied to larger ECUs with dynamic and preemptive scheduling. However it is typical that there are more smaller, single-purpose ECUs than high-performance units in the vehicle network.

The approach cannot reliably detect an intrusion. There may be false positives and false negatives, depending on the particular value of δ. However, an alert may give a strong indication that the system may have been corrupted.

Since this mechanism is orthogonal to the other security mechanisms of the architecture, it can be very well combined with other alert signals, e.g., from classic intrusion sensors as described in the previous Section.

In contrast to binary instrumentation for data flow analysis, this approach can be implemented in the HSM very efficiently with few instructions. No additional computation on the application CPU is performed. The requirements on the HSM can be satisfied even by the smallest EVITA HSM module.

A general problem is the handling of detected intrusions. Raising an alert or inhibiting the use of key material is not without problems in the context of vehicular networks, as functional availability is of very high concern for passenger safety. We cannot provide a solution to this higher level problem, as the according impact is very case specific.

5.4 Related Work

5.4.1 Exploit Prevention Techniques

Many techniques to defend against exploits on personal computers have been proposed, developed, and deployed in common computer installations during the last years. A non-executable stack (i.e., marking memory regions as non-executable) protects against the injection of shellcode [Pes97]. Address Space Layout Randomization (ASLR) introduced an additional burden that protects against the return-to-libc exploiting technique, which was again defeated by Return Oriented Programming (ROP) [SPPG04]. A very recent technique of aligning code during compilation protects against ROP [OBL+10], but was not yet widely implemented. These techniques are slowly beginning to be integrated into today's CPU architecture, operating system memory management, and compilers. Such protections become prevalent in the PC environment, but they are only slowly taken up in the embedded world, e.g., ASLR was only introduced with iOS 4.3 and with Android 4.0, both released in 2011. Similarly, it was just introduced with the latest version of QNX, a popular operating system for vehicle head units. However, as vehicles typically have a lifespan of over ten years, newly introduced exploit techniques likely apply to most vehicles on the streets.

Moving towards dynamic protection methods, runtime data flow tracking using input data tainting has been used. In contrast to dytan [CLO07], the libdft framework [KPJK12] is specifically designed to run at a low overhead and to provide a generic, yet flexible toolkit for taint analysis. The sample libdft-dta

tool, which we evaluate in Chapter 6, shows the applicability in an exploit-prevention context.

TaintExchange was the first approach to demonstrate the feasibility of distributed taints using libdft [ZPK11]. This system instruments all system calls used to exchange data between processes and to communicate over network sockets. In contrast, our approach adds a ruleset, which describes the acceptable behavior of software modules in a declarative and fine-grained manner.

5.4.2 Information Flow Control

The term information flow control originates at policies for large organizations with different clearance levels, i.e., some people can read more documents than others. In [Che09] such clearance labels are introduced already at the middleware layer: he proposed a self-contained .NET based system that propagated those labels among remote execution and data exchange. Another interesting approach to limit sensitive data from leaving local applications was taken by [CC11]. For every process, they run a copy (a shadow process), which is only different in the detail that private data are exchanged for random data. They compare the output of the programs (e.g., of write calls) and find whether the result of the original and shadow process are the same (no private data used) or differ (private data was used in calculating the results). They claim that their approach is up to thirty times faster than taint tracking. While this is an interesting approach to avoid data leakage in enterprises, it offers no protection against platform attacks, which is one of our goals, and which also may lead to privacy leakage. The dynamic analysis of information flow within programs in order to study malware behavior has also been done with the help of emulated environments, e.g., Valgrind [NS05] or QEMU [YSE+07]. While these approaches are well suited to analyze given malware programs, it can hardly be applied to our application domain. There also exists a tool based on the Pin binary instrumentation tool for Windows, which allows the user to trace whether certain input leaves a program via the network [ZJSK11].

5.4.3 Current On-Board Security

In the last two years, the automotive on-board network has come to the attention of security researchers. We have seen that a large effect (i.e., the complete control of a vehicle) can be achieved with relatively common techniques locally [KCR+10] and remotely [CMK+11]. Vehicles have been criticized for broadcasting private data and not performing input validation [RMM+10]. Upcoming Car2X communication demands for higher security levels and introduces hardware security [PBH+08].

Intrusion detection on automotive buses has been discussed in [LNJ08] and [MG09]. In [LNJ08], a specification-based approach that monitors the consistency of frames with the CANopen standard is used in order to exemplify the detection of malicious activity on CAN buses. In addition to scrutinizing the frame layout on the bus, ECU and application specific rules can be used. In the setup they propose, detectors are deployed at every ECU in order to identify the misbehaving nodes. In [MG09] an approach is described, where the sensors are placed at the network and not at every ECU. The approach classifies six different types of sensors at three levels: Packet, Network, and Application. It classifies formality, location, compliance, correlation, frequency, and plausibility sensors. At the packet level, formality and compliance detect malformed network frames and payload. At the network level, the frequency of messages, the correlation between frames, and the location of data (i.e., is a specific frame allowed on a specific bus) are assessed. At the application level, the plausibility of the payload is assessed.

Our approach using distributed dynamic information flow tracking cannot easily be placed in Müter's classification: Data flow tracking is not done at the network level, but also goes beyond plausibility checks at the application layer. It can be seen as an extension of the classification: A usage-control sensor, that would be placed inside individual ECUs in order to enforce strict usage control on the data used by applications. We can thereby prevent data leakage and host-based intrusions. The other sensors of Müter's classification can directly be implemented as part of the distributed IDS system we presented in the first part of this Chapter.

The main publication supporting this Chapter is [SR12].

Chapter 6

Prototypes and Evaluations

> "First get your facts; then you can distort them at your leisure."
>
> Mark Twain

In this Chapter we perform a number of experiments on prototype implementations with which we validate our approaches from Chapters 4 and 5. We have set up different prototypes for the different security mechanisms we proposed.

For key distribution and secure communication protocols, the latency of communication and the increased bus load are analyzed. The protocol performance is shown for the CAN bus. We first measured these protocols for feasibility in two simulation environments and then integrated them with the framework in a stand-alone setup with three computers.

In order to show the current performance and feasibility of the dynamic distributed information flow tracking concept that was introduced in Chapter 5, we analyze the processing times of the prototype implementation with two different taint engines. This prototype was integrated in the prototype mentioned above.

An integration into two real vehicles for a demonstration of 'Active Brake' (Scenario I) was achieved as part of a demonstration event in November 2011, which we describe in the last part of this Chapter.

This Chapter starts with an analysis of the current utilization of a CAN bus in a typical upper class vehicle, in order to show that our approaches for securing communication are feasible even with today's bus loads.

(a) Number of CAN frames per second

Minimum	224
Maximum	265
Mean	237
Std. Deviation	9,1

(b) CAN payload per second (bytes and percent of capacity)

Minimum	1322	21.715%
Maximum	1594	23.891%
Mean	1396	22.307%
Std. Deviation	60	0.91%

Table 6.1: Body-CAN: frame count and payload per second, based on a random sample of 84 seconds. The third column of the right table denotes the estimated bus load for the values, including a protocol overhead of 47 bits per frame (standard 11-bit addressing), and neglecting additional overhead introduced by bit-stuffing.

6.1 Analysis of a CAN bus

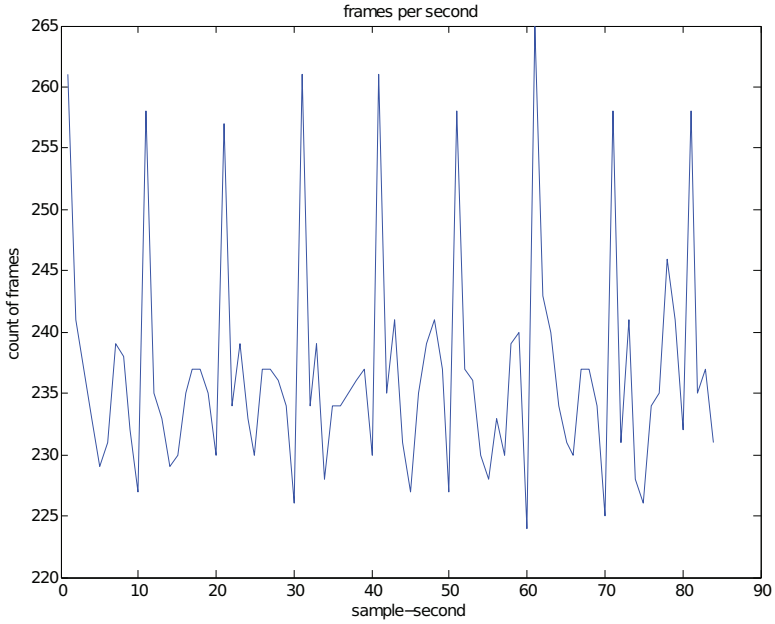
In order to analyze the available bandwidth for the security mechanisms that we suggested, we have performed measurements on a typical vehicle cabin-bus, i.e., the bus that connects convenience and infotainment ECUs inside the cabin. This bus is often referred to as "The Body CAN" or simply "Low Speed CAN". It uses non-terminated wires and low-speed transceivers at 100kbit/s.

We have taken samples from a fully equipped vehicle (model year 2008). As it is equipped with all optional features, we can assume that our measured bus load is approximately the maximum of what is expected for periodic payload on other vehicle's buses as well.

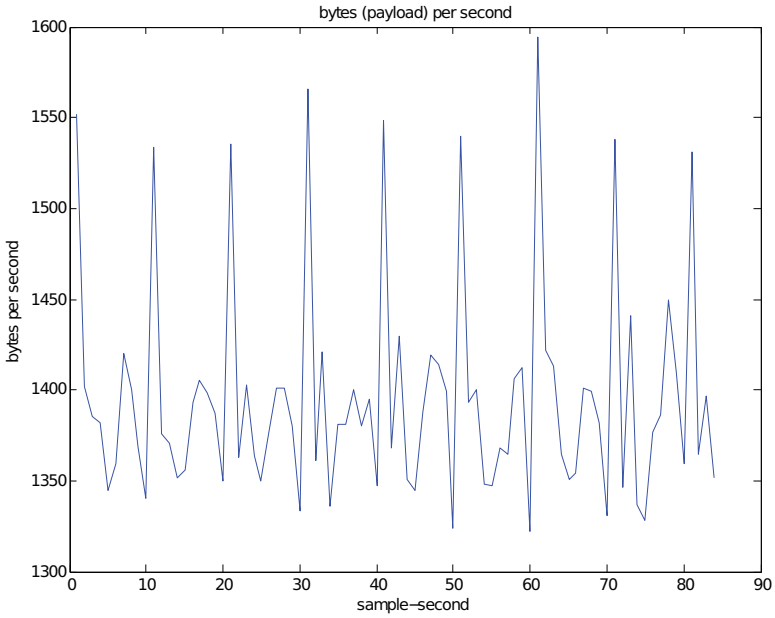
We have observed traffic patterns of periodic communication on the network. Figure 6.1 displays two time series. The first time series shows the number of frames per second. The second graph shows the payload per second that is transmitted. Both time series show a periodicity of around 10s, which is common for on-board networks where a "signal paradigm" persists, i.e., digitized sensor values that are transmitted regularly, and in short intervals.

To our surprise, we saw that the maximum channel load is only at about 22%, leaving 78% of the channel unoccupied. This may be due to pessimistic worst-case scheduling scenarios for realtime constraints. We looked at the specific spread of CAN identifiers in order to obtain a clearer picture regarding the distribution of high and low priority data. A fine-grained histogram showing the observed distribution of CAN identifiers is shown in Figure 6.2. There is only a small cluster of high priority frames, around ID 200 (decimal), in the histogram.

The results of this analysis show that there is indeed a reasonable amount of bandwidth left for security enhancements. Even if all traffic would be secured by two additional frames, i.e., the data volume would be tripled (resulting in 66% bus load), the bus would still not be saturated. The CAN bus simulations of the next section show that the expected signal latency is only slightly increased, even at high bus loads.



(a) CAN frames per second



(b) CAN payload per second

Figure 6.1: CAN frames and payload per second (based on random sample of 84 seconds). We can see a periodicity of 0.1 Hz.

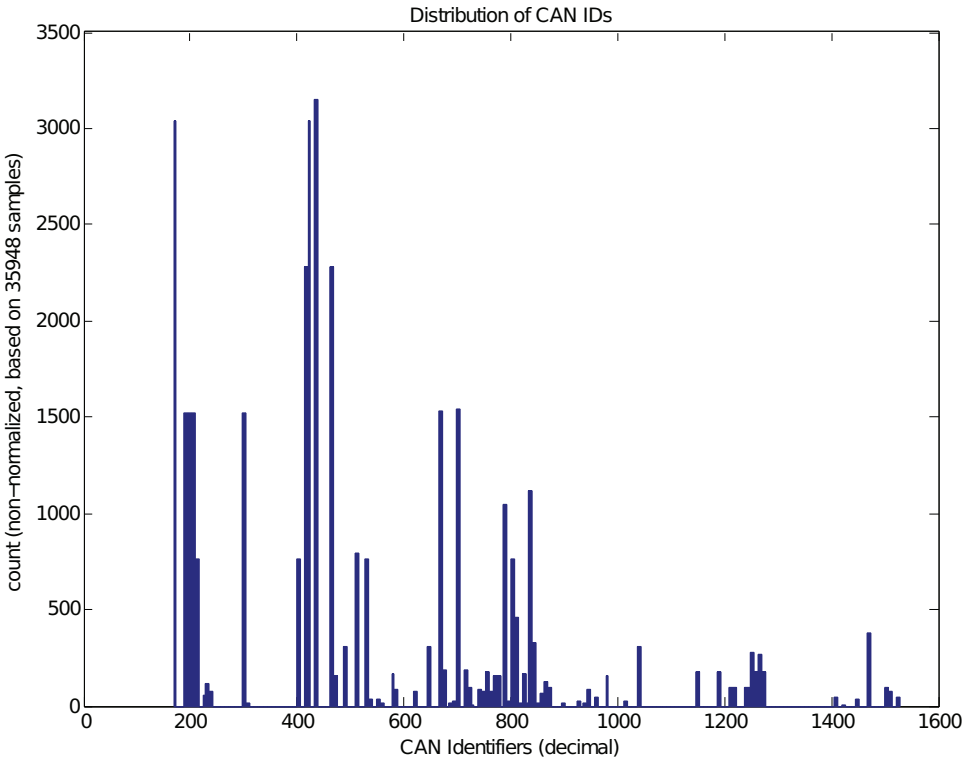


Figure 6.2: Distribution of CAN identifiers in a 200-bin histogram. The count is absolute with a total number of samples of 35948 in 2:32 minutes.

6.2 Protocol Measurements

In the following, the key distribution protocol and the secure communication protocol are evaluated for the CAN bus. For simulating the key distribution protocol (Section 6.2.1), we used the performance specifications of the overall system, down to clock cycles in the HSM. However, for network arbitration, i.e., the media access on the CAN bus, a simplified priority model is used. As key distribution is only done once for every communication group, the network performance is not as important as for secure communication.

For the second part (Section 6.2.2), we used a different simulator to show secure communication using the CAN transport protocol. In this part, the bus arbitration, and especially the additional latency at different bus loads were more important, as secure communication is anticipated for continuous use.

6.2.1 Simulating Key Exchanges with TTool

We modeled the key distribution protocol in a scenario as described in Section 4.1.4. The according sequence chart is found in Figure 4.2 on page 59. The setup comprises the initiator, ECU_1, a key master and the receiver ECU_2, which are all connected to the same CAN bus and equipped with an HSM. This setup is shown in Figure 6.3. The model includes several assumptions about frequency and load of processors and network media. Specifically, the HSM's cryptographic engine is clocked at 100 MHz and requires 60 clock cycles, including scheduler cycles, for the encryption of one AES block as well as for MAC blocks. These are the real-world parameters for our FPGA prototype. The CAN bus is loaded at 30% (as shown to be realistic by our experiments in 6.1) and the arbitration for every individual packet is won with a probability of $p = 0.5$. Messages transmission includes overhead on CAN (39 bits header, 25 bits trailer, 3 bits idle, bit stuffing omitted) and overhead for the segmentation (2 bytes for the first frame, 1 for consecutive frames). Please note that in contrast to the analysis on the current CAN bus we assume extended addressing for increased flexibility, thus the extended overhead. The HSM keys are constructed as described in Section 4.1 and thus carry header fields of 86 bits in addition to the key length of 256 bits and an authentication code of 128 bits. Thus, for the successful transmission of an HSM key structure via CAN-TP, nine frames are sent: $9 = 1 + \lceil (470 - 48)/(7 \times 8) \rceil$.[1] We assume the CAN bus speed is 500kbit/s, as this speed is commonly used for high-speed buses in vehicles.

This system has been modeled with the DIPLODOCUS UML profile [AMAB+06], and simulated with the simulation engine of the TTool system [KAP09]. The DIPLODOCUS profile and its simulation engine explicitly take hardware elements

[1] The first frame offers six bytes of payload; thus 48 bits are subtracted from the HSM key of 470 bits. Consecutive frames offer 7×8 bits of usable payload.

of the system, that is, CPUs, buses, memories, and the HSM as hardware accelerators into account for the simulation.

A complete key distribution cycle including acknowledgements is, under the load and speed assumptions mentioned above, performed within 9 ms on average. While this is not adequate for on-demand establishment in hard real-time applications, it is certainly feasible to perform the key exchange at boot time for all statically known communication groups. The current HSM implementation allows for every ECU to participate in 64 groups, which is sufficient for the needs of current and future applications.

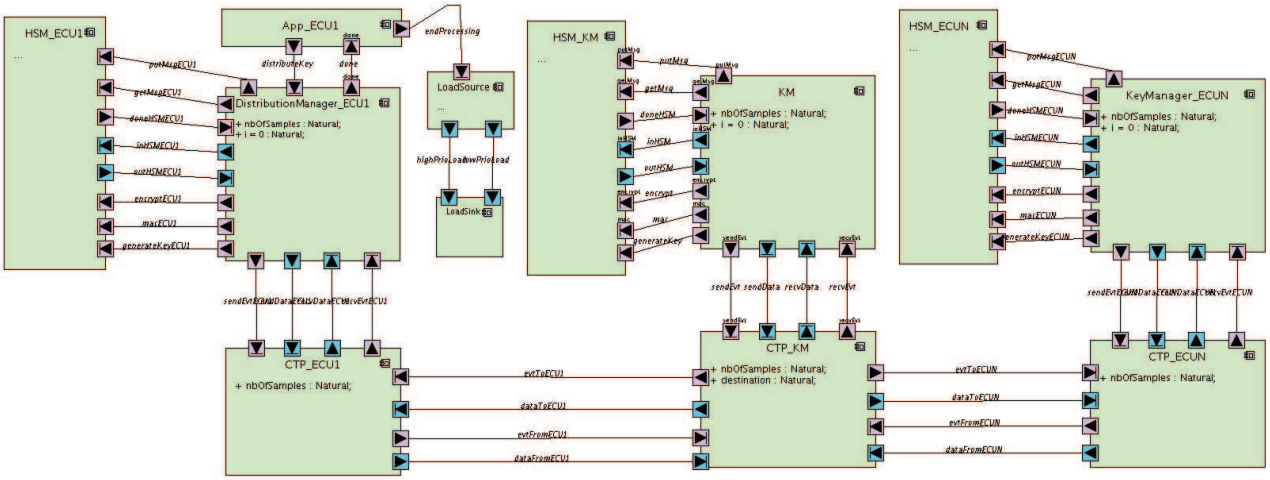


Figure 6.3: TTool simulation model for Key Distribution: Two ECUs and one Key Master node, all equipped with an HSM. All nodes are connected via a CAN bus using the transport protocol. The key exchange is initiated by ECU_1.

6.2.2 Simulating the Secure Transport Protocol

We have also tested our design using a Matlab/Simulink simulation engine, the TrueTIME 2.0 toolkit [CHL+03], which supports the simulation of CAN networks. The simulation setup is shown in Figure 6.4. TrueTIME supports the S-function concept inside Matlab/Simulink, so that our protocol simulation was directly written and compiled in C.

For simulating the transport protocol, we use a more in-depth simulation of the arbitration phase. The setup consists of three nodes, connected by a CAN bus (TrueTime's `networked` setup). One node generates high-priority background noise, which will always be prioritized over our payload. We varied the bus load introduced by this "interference node". The node occupied the bus at random times, using between 0% and 90% of the available slots for sending data at high priority, so it would interfere and take precedence over regular transmissions.

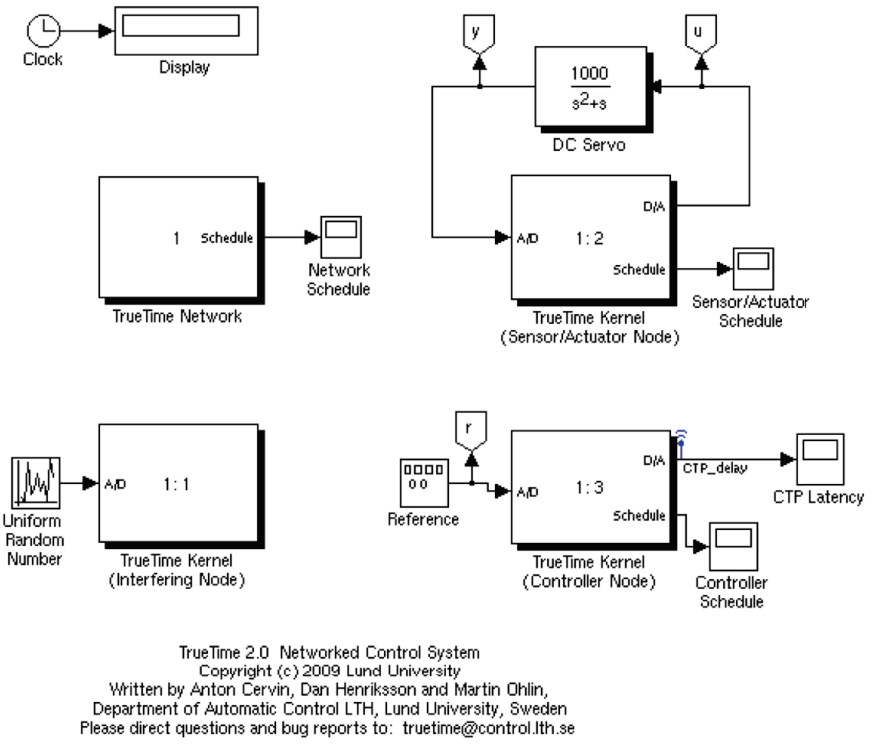
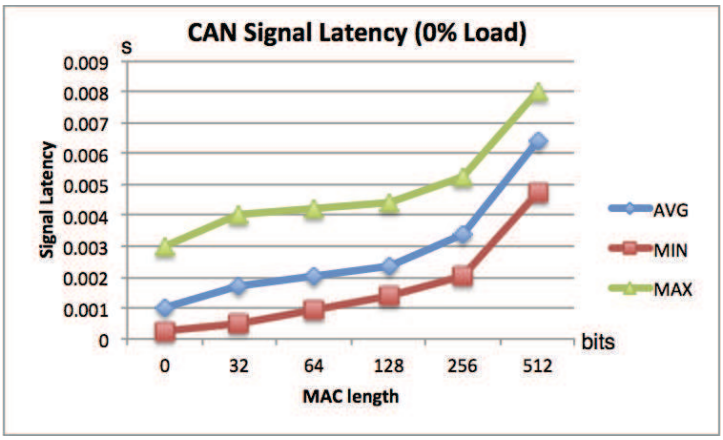


Figure 6.4: Simulink Setup for TrueTime v2.0 simulations: A CAN bus connects a controller node, a combined sensor and actuator node and an interference node (the `networked` setup). The communication is performed in a closed loop: The controller sends step-wise commands to the servo engine (the actuator) and, independently, receives the currently sensed state from a sensor. The servo's response function is $\frac{1000}{s^2 s}$. The communication links were changed to segment data according to the CAN-TP transport protocol, implemented in Matlab S-functions.

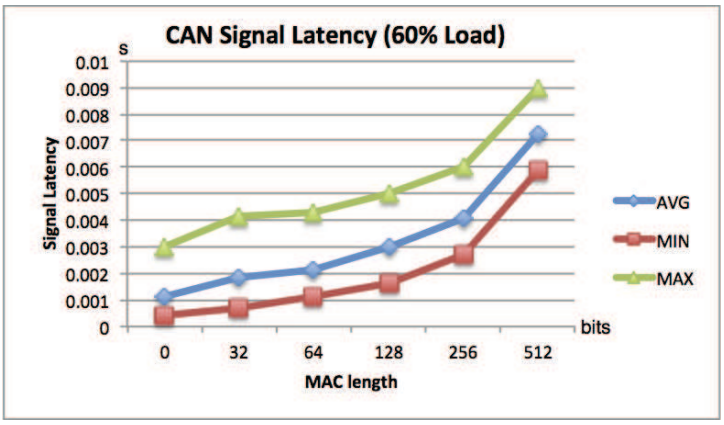
The second and third node model a controller and a sensor/actuator node. For communication between sensor and controller we encapsulate the payload with the secure transport protocol and varied the MAC lengths. You can see the resulting latencies in Table 6.2.

The control system's period of one send/receive cycle of the loop is 10 ms. In every test run, the delays of 100 complete submissions, i.e., one second of simulation time, were sampled.

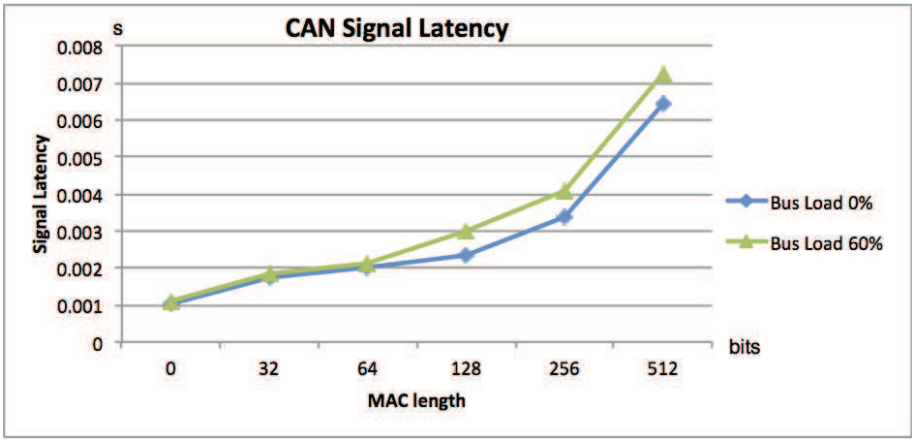
As expected, the latency increases approximately linearly with the number of frames sent (i.e., with the size of the security payload). What we had not expected was that the latency did not increase significantly with the bus load. In Figure 6.5, we show the latency results for bus loads 0% and 60% that only diverge for security payloads larger than 128 bits. A comparison of the average latency for both bus loads is shown in the third graph of Figure 6.5.



(a) CAN Signal Latency at 0% Bus Load



(b) CAN Signal Latency at 60% Bus Load



(c) Comparison of Average Latency

Figure 6.5: Signal Latency for different MAC lengths at 0% and 60% bus load. The difference in signal latency increases only for longer MACs at 60% load. The time is given in seconds and the length in bits.

MAC/bits	0 (SF)	32	64	128	256	512
Minimum	0.0004	0.0007	0.0011	0.0016	0.0027	0.0059
Average	0.0011	0.0018	0.0021	0.0030	0.0041	0.0073
Maximum	0.0030	0.0041	0.0043	0.0050	0.0060	0.0090

Table 6.2: Latency of secured CAN/TP packets with 16 bytes payload in seconds. The CAN bus is charged with higher priority traffic at 60% load. The cumulated processing time is 0.3 ms. The sampled transmissions were repeated 100 times for each MAC length. The first column (SF) represents the latency of a single frame without authentication code.

Discussion In combination with the results found in the analysis in Section 6.1, it becomes clear that additional security payload can be added to signals where needed, even on a CAN bus.

The overhead that is introduced through the transport protocol could as well be saved, if only small fractions of 32 bits are used in combination with at most 32 bits of payload. This would, however, not be suited against replay attacks, as no timestamp or sequence number would be included.

The first column in Table 6.2 shows the transmission latency of a single frame. For the simulation, a processing time of 0.15 ms was assumed for the sender and receiver side (i.e., 0.3 ms are static overhead).

6.2.3 Implementation as Part of the Framework

We implemented the key distribution protocol with the software security framework. On the master side, a new module, the KMM was introduced. It receives keying requests via the EMVY Server-RPC interface, evaluates local policies, and performs import and export of the key according to the protocol (c.f. Section 4.1.4). The session key is distributed to all group members via the Client-RPC interface, where it is imported into the HSM, the key handle being saved for the corresponding group.

As the in-vehicle prototype was, in the end, completely based on Ethernet, we conducted measurements for key distribution latency in the Ethernet setup with all three computers connected via a switch. We found an average latency of over 50 ms. While not prohibitively large, as key distribution only needs to be performed once per drive-cycle, it seemed rather high, especially given the 9 ms that resulted out of our simulations on CAN, and given the absence of any significant delay that was expected for Ethernet. During the investigation of the root cause for this increased latency, we found out that our HSM prototype implementation, which was connected via TCP/IP, added a significant overhead. We conducted a timing analysis of the HSM prototype that allowed us to precisely measure the overhead of key generation, import, and export functions. The results of these measurements can be found in annex B.1.

As a result, the communication and marshalling for the prototypical HSM interface distorted the measurements. A real-world deployment needs to be adapted to different networks and buses. We thus provide no further performance results, but emphasize that even with a latency of over 50 ms, a distribution of sessions keys per drive-cycle is reasonable.

In Annex B.2, code listings give an idea of the programming interfaces used and exposed by the implementation.

6.3 Distributed Dynamic Information Flow Tracking

Applications in our prototype are implemented using a C-style middleware. This middleware establishes communication links to other entities and secures the payload accordingly (UDP/TCP/CAN-TP), as described in Section 3.4. The information flow tracking concept for the vehicle architecture, which we introduced in Section 5.2, uses this communication middleware to enable taint tracking throughout the vehicle network.

Whenever a program reads from I/O channels that are configured as $taint\ sources$, the corresponding data are marked. In contrast to approaches which focus on avoiding run-time attacks (using a single binary tag per byte), we use distinct tags in order to distinguish between different sources.

We used the dytan taint engine [CLO07], which is attached as a Pin tool [LCM+05], i.e., applications run instrumented. Additionally to tagging I/O data at our middleware functions, dytan possesses of an XML configuration file, which lists network streams and files that should be regarded as additional taint sources. Generic system calls such as $read/write$ are instrumented accordingly. Dytan itself is implemented in C++ and allows a large number of distinct tags to be followed in shadow memory, held in an STL map. While this kind of implementation offers great flexibility, we have seen that it performed rather poorly, compared to other engines, as the code injected through binary instrumentation is implemented in C++. However, it has some distinct features (the virtually unlimited number of tags) that were advantageous to show the interest of our approach. We compare it to the performance oriented engine libdft [KPJK12] in the next subsection.

We limited the number of tags to 32, in order to place them into a four byte field to ensure a lower overhead for network propagation of the taints.

We used a standard Ubuntu Linux (v2.6.38) on an i5 processor. We compiled all programs in 32 bit mode and without MMX and SSE instructions, as the taint engine does not support these registers, nor 64 bit operations.

6.3.1 Performance

Instrumenting binaries with additional code incurs runtime penalties, as for each original instruction, additional instructions are injected, thereby slowing down the overall execution. As demonstrated in [KPJK12] with the libdft implementation, an optimization of tag-propagation instructions towards modern processors, which feature multi-stage pipelines and jump-predictions, can result in much higher performance, compared to heavier implementations. Unfortunately, the original libdft implementation only allows to track binary taints, i.e., a "taint or no taint" tag. While this is sufficient in order to detect previously unknown exploits that trigger buffer overflows and similar, it does not offer a tag-space sufficient for system-wide information flow monitoring. However, in order to show the impressive performance of their system, we also conducted experiments using libdft and included them in our results. An alpha version of libdft supporting 8 color taints showed exactly the same performance. This is due to byte-wise operations, which have less execution overhead than bit-wise operations, which require masking and shifting instructions, as detailed in [KPJK12].

We measured the runtime of a modified version of the music player mpg123, which we integrated with our framework in order to send the MP3 ID3 tags to a remote display (see the example in Figure 5.5 on page 87). The runtime includes extracting the ID3 information, the process of marshalling information with taint tags and generating a MAC, as well as decoding the first 100 frames of the MP3 file, corresponding to 2 seconds of music.

	vanilla	nullpin	libdft-dta	dytan
runtime in ms	37	377	1 045	18 925
factor (v)	1	10.2	28.2	511.5
factor (i)		1	2.8	50.2

Table 6.3: Runtime of example audio player application (mpg123) with different instrumentations.

Vanilla execution is the non-instrumented, plain execution of the code. The column "nullpin" relates to code instrumented with Pin, but without executing any hooks (part of libdft's sample tools). Libdft-dta and dytan are the actual tag-propagation engines. We show their overhead compared to vanilla execution (v) and instrumented execution by nullpin (i).

Discussion The performance results show that binary instrumentation results in the most significant part of the execution overhead (factor 10). A performance-oriented implementation of the tag engine itself, such as libdft, only incurs an overhead of factor 2.8. If the instrumentation itself can be rendered more efficiently, the overhead of tracking data throughout execution is

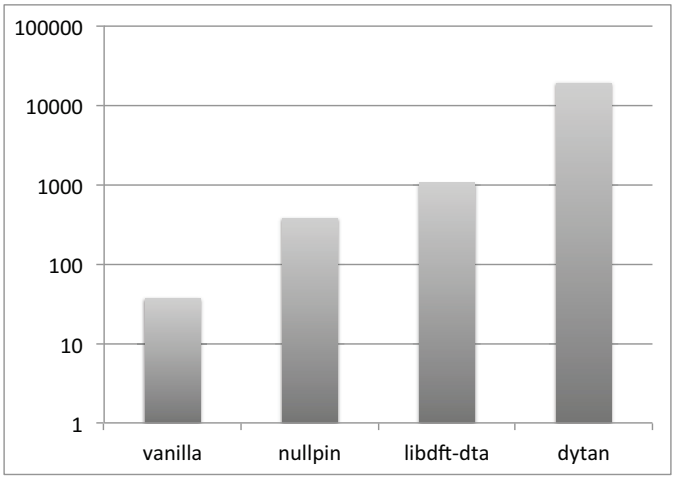


Figure 6.6: Runtime of instrumented code in microseconds on a logarithmical scale. Decoding of the first 100 frames of an MP3 file in i586 instructions. Unchanged code is 'vanilla' runtime, nullpin the instrumentation overhead for PIN, libdft-dta and dytan the corresponding DTA frameworks.

tolerable. While a small number of tags will not be able to construct a full-fledged information-flow system, it can still serve the purpose to limit data usage from certain sources (e.g., user input from media, internet or diagnosis), especially to limit leakage of private information. These figures may seem high, but it should be noted that other approaches to secure applications on the head unit, such as virtualization, incur a significant overhead as well.

6.4 In-Vehicle Prototype

We have integrated the Active Brake scenario that was introduced in Section 3.1.1 in of prototype vehicles. We have presented these vehicles during the final EVITA public workshop and the Car2Car Communication Consortium Forum 2011 at Erlensee, Germany.

The control units of the vehicle network were equipped with EVITA FPGA hardware and used the communication protocols described in Chapter 4.

The brake signal of one vehicle triggered the sending of a DENM warning for the other vehicle, as described in Scenario I on page 39. We created a communication group between the brake sensor, where the pedal depression was sensed, and the brake controller, which triggered the warning message via a sim^{TD} communication unit. The sim^{TD} Communication Control Unit (CCU) was complemented with an EVITA HSM to enhance the performance of cryptographic operations. We show pictures of the demonstrator vehicles and the communication unit in Annex B.6 (p. 178).

6.4.1 The CAN-Ethernet Gateway

The bus system used in today's vehicle is in most cases the CAN bus. In addition to IP communication, we also wanted to show the feasibility of our approach on current systems, and thus implemented a proof of concept for CAN communication within the EMVY framework. Originally, an AUTOSAR port of the EMVY framework was envisaged in the EVITA project, which should have represented the CAN node of the demonstrator setup.

As part of the CAN setup, a CAN to Ethernet gateway was foreseen. We used standard ISO-TP for CAN communication in order to place our application data (e.g., addressing, RPC data, and security payload) on top of it.

Implementation The gateway was implemented using Oliver Hartkopp's SocketCAN [Har11] on a Linux machine. SocketCAN has the big advantage that it uses POSIX sockets with additional `sockopt` arguments in order to establish communication with the ISO-TP transport protocol. It uses address pairs is foreseen in the ISO-TP standard. The use of POSIX sockets facilitated the adaptation of existing network code, which relied on TCP connections and UDP datagram.

We were able to make use of a relay-example of the SocketCAN distribution in order to implement this gateway. The gateway relays the raw ISO-TP payload between CAN clients and a master node on Ethernet. This gateway was used in order to connect both the server-RPC and the client-RPC interface, as well as direct communication between EMVY clients. In Figure 6.7 you can see a generic abstraction of the gateway, which is implemented in a store-and-forward manner for datagrams of up to 4095 bytes, a fixed limitation of the ISO-TP standard.

In order to test the gateway we added SocketCAN also to the EMVY client framework. By doing so, the clients were able to use ISO-TP instead of IP to access services on the EMVY Master, such as key distribution.

Discussion The gateway was foreseen to connect the AUTOSAR implementation of the client framework with the EMVY master node for the final EVITA demonstration. Unfortunately the AUTOSAR implementation of EMVY was never realized and AUTOSAR was demonstrated separately. Thus the gateway was not integrated into the prototype vehicles. As we had already successfully tested the gateway with EMVY on Linux together with SocketCAN and a PEAK CAN adapter, we decided to include it in the implementation Chapter, in order to show the feasibility of the implementation.

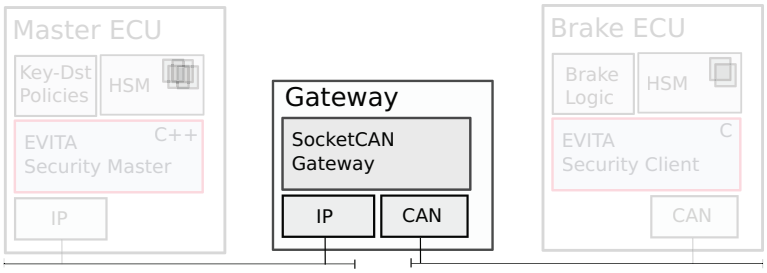


Figure 6.7: The CAN-to-Ethernet Gateway. It can be operated in TCP and UDP-muilticast mode. IP connections are translated to ISOTP on the CAN interface. The payload is not modified but directly relayed.

Chapter 7

Conclusion and Outlook

> "The hardest thing is to go to sleep at night, when there are so many urgent things needing to be done. A huge gap exists between what we know is possible with today's machines and what we have so far been able to finish."
>
> Donald Knuth

This Chapter concludes the thesis with a comparison between the current state of security in vehicular architectures and contributions of the thesis. Thereafter, we critically discuss our approaches for feasibility in a real-world deployment and give an outlook on future research and prospectives for further industrial development of security measures.

7.1 Achievements and Conclusion

We claim that our approaches increase the level of security in vehicular on-board networks by applying cryptography to communication, as well as leveraging runtime security with data tainting techniques for the vehicle network and strict policy enforcement as part of the intrusion detection system. Our prototype and demonstrator implementations are based on a real automotive security framework. We have simulated and tested these approaches for feasibility in current on-board networks. Even on restricted networks, such as the CAN bus, our secure communication approach is viable, and seems to be taken up by industrial development [KB11]. The protocols for key distribution and secure communication have been successfully demonstrated along with Car2X applications

inside two BMW vehicles at the Car2Car Forum in November 2011 in Erlensee, Germany.

The contributions that were presented in this thesis can be illustrated best by the following comparison. A frequently cited publication on embedded security, *"Security in Embedded Systems: Design Challenges"* [RRKH04], names six requirements for security in embedded systems. Based on these generic requirements, we compare the vehicular architecture of today with the enhancements suggested by contributions of this thesis.

The security requirements postulated by Ravi, et al. in [RRKH04] are:

(1) Identification
(2) Secure Network Access
(3) Secure Communication
(4) Secure Storage
(5) Content Security
(6) Availability

Identification, Network Access, and Communication In today's vehicles, a secure identification (i.e., authentication) between entities inside the vehicle is only done at few places: Identifying the driver opening the vehicle, igniting the engine, protecting configuration parameters and saved data such as the odometer inside ECUs, as well as a rudimentary protection of certain diagnosis functions exist today. However, *identification* (1) between components of the on-board network does not exist. Neither are *network access* (2) and *communication* (3) over internal buses secured.

With the approach we presented in Chapter 4, we have introduced an integrated solution for securing communication: It comprises key distribution and a secured transport protocol and enables the authentication and *identification* (1) of nodes between each other. While the physical bus access cannot be restricted, the data exchanged over the network for accessing vehicle services are restricted based on using the agreed session key material, thus providing *secure network access* (2). The communication itself is protected by MACs and thereby provides *secure communication* (3) on the on-board buses. Additionally a timestamp in the protocol offers protection against replay attacks on demand. We have addressed the problem of costly solutions by using symmetric cryptography in an asymmetric manner: Usage controlled key material, where the actual enforcement is embedded in the HSM based on the key's flags. Moreover, we have addressed 1:n group communication as used for communication in today's vehicles. This aspect had been set aside by many previous approaches for secure on-board communication, which aimed at 1:1 communication as used in ordinary computer networks. We also proposed a solution for vehicle maintenance issues dealing with cryptographic key material, a real-world aspect that is of primary concern for actual deployment, as we learned from industrial feedback.

Storage, Content Security and Availability In today's vehicles, only certain parameters of the vehicle, like the Vehicle Identification Number (VIN) are protected in such a way that they can only be written once (burn-in). Some functions are protected through access control measures, e.g., by a two byte code to access the diagnosis mode of an ECU. However, codes and passwords provide protection only to a certain extent, as for example a two byte code can be broken by brute force (trial and error) rather easily. Additionally, data are usually not stored encrypted, making for example the firmware prone to reverse engineering attacks.

The HSM that was used in this thesis provides an internal key store for cryptographic material. We defined a lightweight protocol (details in Annex B.2.4) that makes use of it and that permits us to provide *secure storage* (4) for arbitrary data.

Along with secure storage, our approach applies fine-grained policies and enforces these through dynamic information flow tracking and a network-wide propagation. This assures that information is only used for the intended purpose, i.e., we also fulfill the requirement *content security* (5). We presented our approach towards fine-grained data usage control for privacy and security aspects in Chapter 5.

The last requirement, *availability* (6), is certainly important for the automotive domain. While the focus of this thesis was not on availability from a safety perspective, we showed how, for example redundant key master servers and intrusion response, might prevent a single point of failure from endangering safety critical functions of the vehicle. Distributed systems have created many solutions as far as availability, coordination, and concurrency are concerned. These can constitute directions for future research in order to make distributed automative system both robust and secure.

Additional Requirements: Privacy and Attack Mitigation Since the publication of [RRKH04] in 2004, two important requirements have become apparent: the privacy of data and attack mitigation for embedded systems. Since then, mobile phones are routinely compromised ('rooted' or 'jailbroken' in hacker jargon), both locally and remotely, and applications secretly access and transmit personal and private data.

Our dynamic information flow tracking approach provides a measure against runtime attacks (e.g., software exploits) and the abuse of data. With our approach, we specifically embrace the execution of software of unknown pedigree and with uncertain trust: Rule-based policies are applied for controlling the software and the use of data also beyond the borders of the ECU where the code is executed. We propose a rule-based policy that enhances the 'who with whom' with the *kind and provenance* of data that is exchanged, i.e., the origin of data represented by the taint tag, secured by a MAC. This allows for a fine-grained

enforcement that comports with model-driven development, which is prevalent in vehicle engineering.

Discussion While our solutions to secure the onboard architecture of vehicles were implemented and evaluated with a prototype demonstrator, a deployment in mass production has much tighter requirements. Cost is a major factor in the automotive world: If vehicle components cost $50 more (imagine HSM chips in multiple ECUs), and the vehicle is sold ten million times, this already sums up to half a billion lost profit. Customers are also not willing to pay for security functionality, as they expect the product to be secure as-is, or do not understand the issues at hand sufficiently.

While cost is one concern, the integration into communication stacks and the limited processing power of current components is another. We have shown that the communication overhead for secure communication protocols is limited. Nonetheless, an additional latency is introduced.

Integrating cryptographic protocols with automotive operating systems is not trivial, as many different systems exist, for example OSEK, QNX, VxWorks, or even Windows Embedded Automotive. Low-cost components like sensors often do not have a real operating system, but use a static communication stack combined with a state machine. The AUTOSAR approach with a standardized middleware addresses this problem. However, a wide deployment for low-cost components is still a distant prospect.

The information flow tracking approach using binary instrumentation has the big advantage that it can be used on unknown program code. On the other hand it has a big influence on the execution performance of programs. Naturally, source-to-source compilation for information tracking, as illustrated by the frameworks GIFT [LC06b] or DIVA [XBS06], provides a much better performance (between 5% and 50% overhead), but rely on the fact that the source code needs to be available for recompilation. This is not the case for third party development in open execution environments or application stores. Better performance could be reached by dedicated hardware with integrated taint tracking, or a recompilation of intermediate code, as for the Java runtime or inside a virtualization environment.

Trusted platforms and secure on-board communication leverage the trust needed for received Car2X communication, in particular if actions need to be taken on received data. Still, a single successful runtime attack at the Car2X radio unit, which evaluates internal information and triggers external alerts, is sufficient to compromise the overall system. This is a general problem, as no end-to-end security—from the sensor of one car to the actuator of another car—is achievable without costly asymmetric cryptography and great efforts in harmonization between manufacturers. Further hindrances for world-wide deployment can be

different implementations, as US, European and draft standards in Asia are not interoperable, and export restrictions for cryptography.

7.2 Outlook on Future Research and Development

We have seen that the vehicle domain has not taken security measures into account for the overall on-board architecture. Security measures were only applied at certain exposed interfaces, e.g., the radio signal of a key fob to unlock a vehicle uses some distance-bounding challenge response algorithm. While a complete protection of all on-board units and communication is not realistic for the coming years, it is clearly one of the topics that currently see a lot of development.

Secure On-Board Communication Currently, not even critical functionality seems to be properly secured. Break-in tools like [Jac12] are freely sold. They connect to network cables inside the door crack that can be grasped from the outside, deactivate the alarm of the vehicle, and roll down its windows. This makes the lack of secure communication in today's vehicles blatantly obvious.

The AUTOSAR release v4.0 introduces APIs for cryptographic services as part of the specification [BFWS10], which facilitates the use of cryptographic security in software. What it does not supply, is yet a common secure communication channel between units running AUTOSAR. We were pleased to see that a protocol, similar to the secure transport protocol for CAN, which we proposed, was about to be implemented for a real product as part of an AUTOSAR stack [KB11].

A shift towards Ethernet and IP has been made for certain parts of the on-board architecture [Bru10, Jon10]. Further use of Ethernet in the entire on-board network is currently researched [GHM+10]. With higher bandwidth and larger packet size, the use of IP offers much greater room for security solutions. While being more costly, it allows for established and well tested and proven security solutions from classic networks, such as IPSec, to be applied. According to [Mai12] the introduction of Ethernet for dedicated networks is planned is anticipated for 2013, and Ethernet as the vehicle's backbone network for 2018.

Intrusion Detection and Response Detecting misbehaving nodes and malformed network packets is easier than in normal network environments, because a vehicle is a rather static system with traffic patterns that do not change significantly during runtime, compared to networked computers. Intrusion detection has been investigated for the on-board network [LNJ08, MG09, MA11], and constitutes a part of this thesis and of the EVITA project [WWZ+10]. Future

systems may use unsupervised machine learning techniques that became popular for common IDSs. An aspect that has not been assessed adequately is intrusion response: What is an adequate behavior to a response, which has been detected with a certain confidence. Can parts of the vehicle network be shut off? Wouldn't that open new doors for denial of service attacks? What about notifications: A warning *"you have been hacked"* would certainly irritate the driver. A publication from 2008 [HKD08a] provides a classification of possible responses, up to haptic driver alerts. While multi-modality is used for many vehicle functions, we are sceptic that a 'cyber attack' should put force on the brake pedal or the steering wheel, as suggested. The question of how to react sensibly and sensitively to detections is, in our eyes, still an unsolved problem that is not easy to answer—especially for computer scientists. Research results on HMI design and ergonomics should be considered adequately.

Dynamic Information Flow Tracking Monitoring for which purpose data is used goes beyond simple access control. While DIFT has been used for some time in malware analysis, its runtime usage is quite new, and has only recently been applied to embedded and distributed systems [EGC+10, ZPK11]. This technique is particularly interesting for the automotive world, where embedded software is shifting from in-house development towards "application stores" in the infotainment domain—similar to what was has been seen for smartphones in recent years. Renault will open their R-Link AppStore to developers this year, which is scheduled for November [Lut11]. While a strict review process, as enforced by Apple (see Sect. 2.3.2), provides a certain level of security, it can never provide full assurance. Applications can still slip through the review's monitoring [Sli12], and software faults in applications can lead to runtime attacks as shown in Section 2.3.3.

A DIFT approach similar to ours can be applied to unknown applications, and thereby prevent attacks on the driver's privacy such as [Sli12]. The major problem to solve is performance, because binary instrumentation shows a significant overhead, even when small-footprint taint engines are used. A solution to this problem may be a combined approach, in which trusted functions and well validated code is exempt from such fine granular runtime monitoring. We are currently working on a feasibility analysis of an approach that combines purely middleware-based tracking (that has little to no overhead) with binary instrumentation in a vehicle-wide environment.

In the long run, malware may adapt itself to such techniques. Intrusion detection systems can always be circumvented by some means. For instance we see viruses and worms today that can detect whether they are executed in a virtual environment. This is also true for dynamic runtime analyses. Egele et al. emphasize this in their survey on dynamic malware analysis techniques [ESKK08]: *"If information flow techniques become prevalent in analysis tools, it can*

be expected that malware authors will come up with mechanisms that try to circumvent such systems".

Hardware Security One of the main building blocks within the EVITA project was the Hardware Security Module. It has not only been used to secure on-board communication, but was also integrated with a sim^{TD} communication unit, in which it improved the communication stack with hardware accelerated ECC operations[1]. As a follow up of the FPGA prototype, the PRESERVE EU project [PRE14] will develop an ASIC based on the results of EVITA. A prototype to prepare the final Application-Specific Integrated Circuit (ASIC) is envisaged to be used in the final phase of the French FOT SCORE@F in 2013.

The SHE hardware security module features an approach similar to the EVITA HSM. A comparison is given in [WG11]. It also provides secure boot and hardware acceleration for symmetric cryptographic algorithms. Its specification was agreed between a number of OEMs and automotive suppliers, the HIS consortium. The EVITA partners Infineon and Fujitsu have announced plans to start the production of ASIC chips for automotive HSMs by 2014 [SS11]. Infineon also envisages an integration with their TriCore line of automotive microcontrollers like the TC1797 used in EVITA.

Car2X Communication For most people 'talking cars' and autonomous vehicles are still a vision much in the future. While we will certainly not have Car2X communication in the next two years, there already exist preliminary standards from IEEE and ETSI. Even interoperability and conformance tests for early products have been conducted under an official mandate of the European Community [Eur09b].

We are certain that Car2X communication is going to be part of tomorrow's vehicles. The communication link will be secured by cryptographic signatures, and an on-board HSM is going to be required by the standard: For reasons of security, e.g., key and certificate storage, and also for reasons of performance because pure software solutions cannot handle the number of verifications per second that is anticipated.

A growing awareness for attacks on the on-board network [CMK+11, KCR+10] are strong indicators that security will play an important role in the design of future inter-vehicular interfaces and in on-board networks.

[1] In fact, sim^{TD}'s RSA certificates were not used, but ECC pseudonym certificates were used to sign data that was exchanged between the two demonstrator vehicles.

Appendix A

Résumé Étendu — Français

Contenu

A.1 Introduction 120
A.1.1 Electronique embarquée 120
A.1.2 Motivation 121
A.1.3 Description du Problème 122
A.1.4 Récapitulatif et Plan de la Thèse 123
A.1.5 Organisation de la Thèse 126
A.2 État de l'Art 128
A.2.1 Les Systèmes Véhiculaires Actuels 128
A.2.2 Techniques pour Sécuriser l'Exécution de Logiciels 135
A.2.3 Conclusion 138
A.3 L'Environnement et les Composants 140
A.3.1 Module de Protection Matériel 141
A.3.2 Vers une Architecture Embarquée plus Sécurisée . 143
A.3.3 Scénarios 145
A.4 Sécurité des Communications 148
A.4.1 Distribution de Clés dans le Système Embarqué . . 148
A.4.2 Communication Securisée sur le Bus CAN 150
A.5 Sécurité Dynamique des Plate-formes 153
A.5.1 Suivi de Flux Dynamique d'Information 153
A.6 Conclusion et Perspectives 157
A.6.1 Perspectives en recherche et développements futurs 158

Résumé

L'informatique embarquée est maintenant devenue partie intégrante de l'architecture réseau des véhicules. Elle s'appuie sur l'interconnexion de microcontroleurs appelés ECUs par divers bus. On commence maintenant à connecter ces ECUs au monde extérieur, comme le montrent les systèmes de navigation, de divertissement, ou de communication mobile, et les fonctionnalité Car2X. Des analyses récentes ont montré de graves vulnérabilités des ECUs et protocoles employés qui permettent à un attaquant de prendre le contrôle du véhicule.

Comme les systèmes critiques du véhicule ne peuvent plus être complètement isolés, nous proposons une nouvelle approche pour sécuriser l'informatique embarquée combinant des mécanismes à différents niveaux de la pile protocolaire comme des environnements d'exécution. Nous décrivons nos protocoles sécurisés qui s'appuient sur une cryptographie efficace et intégrée au paradigme de communication dominant dans l'automobile et sur des modules de sécurité matériels fournissant un stockage sécurisé et un noyau de confiance. Nous décrivons aussi comment surveiller les flux d'information distribués dans le véhicule pour assurer une exécution conforme à la politique de sécurité des communications. L'instrumentation binaire du code, nécessaire pour l'industrialisation, est utilisée pour réaliser cette surveillance durant l'exécution (par data tainting) et entre ECUs (dans l'intergiciel).

Nous évaluons la faisabilité de nos mécanismes pour sécuriser la communication sur le bus CAN aujourd'hui omniprésent dans les véhicules. Une preuve de concept montre aussi la faisabilité d'intégrer des mécanismes de sécurité dans des véhicules réels.

A.1 Introduction

Cette annexe donnera au lecteur un récapitulatif sur les contributions de cette thèse. Nous allons discuter des approches qui ont été utilisées pour résoudre les défis actuels en matière de sécurité des véhicules automobiles sur les réseaux de bord.

Nous commençons par introduire le domaine d'application et les objectifs de cette thèse ainsi que ses contributions en vue de résoudre les problèmes actuels dans le domaine de la communication automobile intégrée, dans laquelle les véhicules sont devenus *"des ordinateurs sur roues."* [RH05].

A.1.1 Electronique embarquée

Durant les dernières décennies, les véhicules ont été équipés d'un nombre croissant de commandes électroniques. Sans électronique, les véhicules d'aujourd'hui ne seraient plus en mesure de se conformer aux normes d'émissions actuelles et de satisfaire les attentes du conducteur en termes de confort et de divertissement. Cette électronique est devenue un élément essentiel de l'architecture de l'automobile. Elle s'organise sous forme d'un réseau interne de microcontroleurs,

appelés Electronic Control Units (ECUs), *boîtiers de contrôle électroniques*, ou plus simplement calculateurs en français. Un ECU peut, par exemple, faire partie d'un système de divertissement ou d'une interface homme-machine interagissant avec le conducteur, ou compléter des systèmes techniques et mécaniques. Chaque ECU est connecté à un bus, de même qu'à un certain nombre de capteurs et d'actionneurs qu'il commande. Typiquement, les mesures des capteurs sont utilisées en boucle fermée pour commander des actionneurs. Nous donnons un aperçu sur la façon dont l'électronique des véhicules et les réseaux embarqués sont développés dans le chapitre 2.

A.1.2 Motivation

En plus du capteur et du contrôle des boucles qui font partie intégrante des véhicules d'aujourd'hui, l'automobile est devenue une entité avec diverses interfaces internes et externes. Le véhicule se connecte par exemple à Internet pour différents services en ligne. Il permet de diffuser des contenus multimédias dans toutes sortes de formats et en provenance de toutes sortes de réseaux physiques ou de supports connectés.

Aujourd'hui, il existe un écart important entre ce qu'il est possible de réaliser en termes de communication inter-véhiculaire et la confiance nécessaire vis-à-vis des données reçues, notamment du fait de vulnérabilités potentielles dans les réseaux véhiculaires. Les fonctionnalités logicielles implantées s'appuient notamment sur les informations envoyées par d'autres véhicules. Ceci peut directement influer sur le comportement du véhicule, par exemple, dans une situation de freinage d'urgence. Bien que l'information peut être certifiée et sécurisée comme provenant d'un véhicule donné par une signature cryptographique et un certificat de clé publique, ceci ne peut suffire à donner l'assurance que l'information est authentique et n'a pas été modifiée entre la génération et la vérification de la signature. En particulier, *nul n'est en mesure de garantir l'exactitude des données transmises*, notamment parce qu'un ECU ayant traité ces données dans le véhicule expéditeur peut avoir été compromis et pourrait générer de fausses données qui seraient avant que les données était cryptographique signé.

Des Logiciels Téléchargables De plus en plus de plates-formes d'exécution, surtout dans le monde du mobile, permettent au propriétaire d'installer des applications personnalisées. Dans les premiers temps des smartphones, par exemple, aucune sécurité n'était intégrée à la plate-forme de Palm ou au système d'exploitation Windows Mobile. Avec un nombre croissant de menaces telles que les virus et les worms ciblant les plates-formes embarquées, la sécurité joue maintenant un rôle majeur pour le code exécutable d'un programme téléchargeable. Nous assistons à un mouvement similaire dans le monde de l'automobile, où une personnalisation après-vente des fonctions électroniques est maintenant disponible pour les clients. Cette personnalisation va de l'installation d'applications

météorologiques sur l'ordinateur de bord aux applications mobiles sur les smartphones connectés accédant auz fonctions du véhicule. Renault a par exemple prévu le lancement en Novembre 2012 du magasin d'applications en ligne R-link, basé sur Android App [Lut11].

Contrairement aux systèmes informatiques communs, les systèmes automobiles sont très critiques en terme de sécurité et ont donc besoin d'une protection adéquate en matière de sécurité, surtout avec l'apparition d'une multitude de nouvelles surfaces d'attaque. D'autre part, la plate-forme du véhicule est un environnement intégré : les contraintes de coûts, de latence et de fonctionnalité ont été jusqu'à présent les principales préoccupations. Le coût, par exemple, est la principale raison pour laquelle la cryptographie asymétrique ne peut pas être déployé sur des unités de contrôle typiques ou sur des capteurs électroniques.

A.1.3 Description du Problème

Dans ce système formé de plusieurs sous-systèmes, l'échange de données entre les domaines diminue les possibilités de segmentation des réseaux. Dans les véhicules de la prochaine génération, les applications téléchargées peuvent se connecter au réseau embarqué pour accéder aux informations et fonctions de la voiture.

Contextes d'Attaque

Nous nous attendons à de nouvelles menaces à l'égard des systèmes automobiles du véhicule par le biais de trois nouvelles interfaces très exposées. Les *contextes d'attaque* sont plus larges que des vecteurs d'attaque particuliers. En plus des interfaces existantes, telles que le connecteur OBD-II ou les interfaces sans fil mentionnés au début de ce chapitre, ces trois nouveaux contextes d'attaque créent une situation complètement nouvelle dans le véhicule. Plus précisément, ces contextes sont :

1. Canaux de communication véhicule à véhicule ou véhicule à infrastructure, utilisés pour la sûreté et les fonctions de confort.
2. Appareils portables des passagers ayant accès à des interfaces bien définies.
3. Plate-forme d'exécution à l'intérieur du véhicule qui peut exécuter du code étranger, par exemple suite à l'achat par l'utilisateur d'applications [Lut11, MTV09, GEN09] et sur demande pour les services ITS [Kom10].

La figure A.1 illustre dans quelles parties du véhicule ces interfaces plus ou moins ouvertement accessibles sont situées. En particulier l'exécution de code

d'origine inconnue et non sécurisé pose un problème qui ne peut être résolu par une technique de protection unique. Dans la section 2.2, nous présentons différentes techniques, et les classons par rapport à la complexité et à la sécurité. Nous décrivons aussi comment certaines d'entre elles peuvent être combinées.

Scénario Pour le moment les trois contextes d'attaque sont peu connecté. Bien que les applications à l'intérieur du véhicule aient déjà maintenant plusieurs sources de données distribuées et fournissent leurs données à d'autres nœuds dans le réseau du véhicule, elles ne nécessitent pas l'accès en ligne. À l'avenir, ces sources puits de données seront encore plus distribuées, c'est-à-dire que les applications auront accès à des services en ligne sur des serveurs back-end, des services ITS ou des services Internet. Les frontières entre ces services diminuent et des applications de seconde monte (after-market) sont intégrées aux services internes du véhicule. Ainsi, les trois contextes d'attaque grandiront ensemble, posant des menaces potentielles à tous les domaines réseau internes.

Problématique Dans le scénario, les acteurs sont fortement interconnectés et peuvent même changer de rôle. Plusieurs domaines sont mis en oeuvre, dont certains sont très limités (le véhicule sur le réseau de bord) et certains sont accessibles au public (services Internet), mais ils ne suivent pas une architecture ou des mesures de sécurité communes. À l'heure actuelle, certaines mesures de sécurité ne sont basées que sur l'isolation (par exemple, la séparation physique des réseaux du véhicule). D'autres sont basées sur des mesures cryptographiques et de certification (authentification pour les serveurs Internet) ou des politiques de sécurité (règles de firewall). Le scénario ne peut donc pas facilement être évalué dans son ensemble avec les techniques de sécurité classiques, telles que la cryptographie, les politiques de contrôle d'accès ou les TPM. En raison du coût et des exigences fonctionnelles, on ne peut applique des techniques de sécurité quelconques sans justification.

Les composants électroniques d'un véhicule constituent naturellement un système distribué. Par conséquent, la sécurité doit être considérée dans son ensemble, et non isolément pour un ECU en particulier. Comme les flux d'information existent également dans différents domaines, une solution de sécurité doit être conçue de façon conforme aux exigences de sécurité, aux scénarios, et aux applications inter-domaines.

A.1.4 Récapitulatif et Plan de la Thèse

L'objectif global de cette thèse est de réduire le risque d'attaques provenant des trois contextes d'attaque et de leur combinaison. Un problème central non résolu des réseaux embarqués d'aujourd'hui est de maintenir et d'attester de

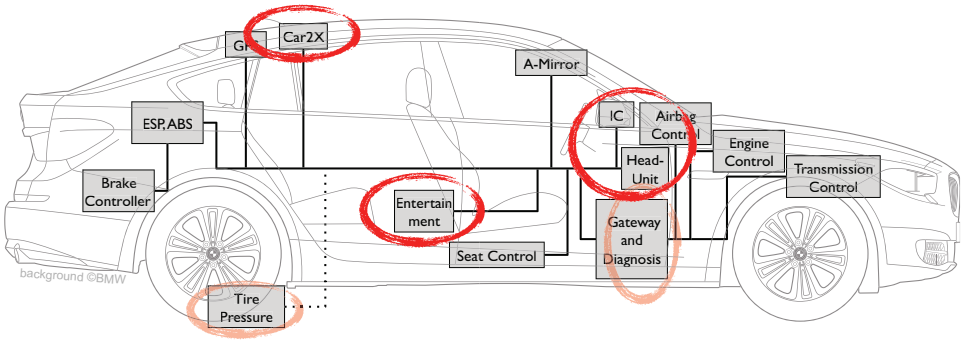


Figure A.1: Nouveaux contextes d'attaque des véhicules connectés : Car2X, les appareils mobiles et les applications téléchargeables sont les principales interfaces exposées. Interfaces de diagnostic et interfaces sans fil (par exemple, capteurs de pression sans fil) sont également sans protection spéciale. Le schéma du véhicule affiché en arrière plan est propriété de BMW.

l'authenticité et de l'intégrité du réseau interne du véhicule entre les composants. Celui-ci est beaucoup plus exposé qu'il y a quelques années.

Alors que les trois contextes d'attaque semblent, à première vue, orthogonaux, une combinaison de techniques peut nous permettre d'atteindre l'objectif visant à prévenir et à limiter l'impact des attaques et garantir la confidentialité du système. Le troisième contexte d'attaque exige explicitement la protection de la vie privée et la sécurité de l'exécution. Par exemple, un code étranger pourrait être chargé sur un microcontrôleurs du véhicule et accéder à des informations personnelles, telles que les informations de paiement ou aux services de navigation et à l'historique du véhicule. Aujourd'hui, nous voyons déjà des exemples de vol de données ciblant les appareils mobiles : les applications peuvent envoyer secrètement des données de localisation pour les entreprises de publicité, ou prendre en charge le périphérique lui-même en vue de commettre des paiements ou des appels frauduleux. Ce type d'infraction peut aussi se reproduire pour les plates-formes véhiculaires, qui actuellement s'ouvrent aux mêmes types d'applications.

Nous avons contribué à la conception et aux réalisations du projet européen EVITA du 7ème programme cadre. Nous décrivons l'environnement du prototype d'EVITA qui a été utilisé dans le cadre du travail de cette thèse dans le chapitre 3. En particulier, la plate-forme logicielle et le module de sécurité matériel sont des éléments importants pour nos implantations. Comme nous ciblons spécifiquement le contexte véhicule, nous soulignons certaines contraintes spécifiques et les conditions préalables pour la communication de l'automobile et de plates-formes d'exécution.

Communications Une approche classique de la question de la validité des données reçues est la cryptographie. Dans le véhicule, il n'y a actuellement aucune technique possible pour résoudre ce problème : les ECUs n'offrent pas de puissance de traitement suffisante pour des opérations cryptographiques supplémentaires et les bus de communication ne permettent pas de définir des protocoles comportant des messages significativement plus longs, une exigence pour introduire des signatures numériques ou des codes d'authentification. Une contrainte supplémentaire est posé par des exigences de temps réel pour certaines applications automobiles.

Plate-forme Alors que les mécanismes cryptographiques peuvent assurer la fraîcheur, l'authenticité et l'intégrité des données, le récepteur doit avoir la certitude que la plate-forme de l'expéditeur est dans un état de confiance et non compromise. Cela signifie que l'environnement d'exécution et le logiciel qui a généré les données reçues n'ont pas été modifiés par un tiers. Dans les systèmes informatiques personnels, cette question est abordée par une technique appelée attestation à distance, dans laquel un module matériel sécurisé comme une puce TPM mesure l'intégrité du logiciel. Bien que certains problèmes, par exemple en ce qui concerne la vie privée ou la fraîcheur d'attestation peuvent survenir [CGL+11], nous considérons que l'approche d'attestation de sécurité de la plateforme est viable pour un véhicule. Pour des raisons techniques et pour des raisons de coût (par exemple, la nécessité d'utiliser le chiffrement symétrique), une attestation à distance des composants dans le véhicule pour sécuriser la communication avec d'autres véhicules (communication Car2X) est hors de question. Pour la communication externe Car2X, les approches employées s'appuient soit uniquement sur la cryptographie sous diverses formes (asymétrique [GFL+07], de groupe [GBW07], temporisée [HL06]), soit sur des informations de réputation dynamique [Ray09] comme indicateur supplémentaire.

Contributions de la Thèse

Nous avons développé des techniques et des protocoles pour les réseaux véhiculaires embarqués qui peuvent assurer l'authenticité et la fraîcheur des données (sécurité), ainsi que la protection effective des données (sécurité et confidentialité) au sein du réseau du véhicule et à l'intérieur des ECUs.

1. Spécification et Conception
    - Un protocole embarqué pour la distribution des clés cryptographiques.
    - Un système de détection d'intrusion (IDS) distribuée pour les réseaux et plates-formes d'exécution véhiculaires, basés sur le suivi des flux d'information dynamique (DIFT).

- Une approche basée sur des règles de DIFT pour décrire les flux d'information autorisés.

2. Simulation et Evaluation

   - L'évaluation de la latence et de taux d'erreur du protocole de transport sécurisé sur des noeuds CAN simulés.
   - La performance globale de la distribution des clés sur un bus CAN simulé et sur un prototype Ethernet.
   - L'évaluation du prototype de l'approche de suivi de flux d'information distribué.

3. Implantation du Prototype

   - Distribution des clés, intégrée avec le module HSM et l'intergiciel EMVY.
   - Communication sécurisée pour les bus CAN et Ethernet, intégrée avec le HSM et l'intergiciel EMVY.
   - Une passerelle CAN–Ethernet.
   - Un système de détection d'intrusion pour le véhicule, intégré à l'intergiciel EMVY.
   - Un système DIFT distribué, basé sur des moteurs de marquage de données locaux utilisant l'instrumentation de code compilé et la propagation de marques par des liens de communication sécurisés.

A.1.5 Organisation de la Thèse

Cette thèse est divisée en sept chapitres. Dans ce résumé en français nous allons donner un bref aperçu du contenu de chaque chapitre. Le lecteur se reportera à la version anglaise pour plus de détails.

Suite à l'introduction, le chapitre 2, *Etat de l'Art*, donne un résumé complet des pratiques industrielles actuelles et des approches de recherche à la fois dans le domaine des véhicules et d'autres environnements informatiques embarqués tels que les téléphones mobiles ou des ordinateurs. Dans le chapitre 3, *Environnement*, nous présentons les scénarios que nous avons choisis pour illustrer les approches développées dans cette thèse. Nous décrivons un modèle d'attaquant, l'environnement d'expérimentation des véhicules, et le projet européen FP7 EVITA, dans lequel la plupart des travaux de cette thèse ont été réalisés.

Cette thèse se poursuit avec les deux chapitres principaux : dans le chapitre 4, *Communication Sécurisée*, un protocole de distribution de clés dans les environnements automobiles ainsi qu'un protocole pour établir une communication sécurisée sur le bus CAN sont présentés. Par la suite, le chapitre 5, *Securité des*

Plateformes Dynamique, évalue la sécurité globale dynamique de la plate-forme automobile : une approche qui est basée sur le suivi de flux d'information dynamique pour sécuriser l'exécution. Ces techniques permettent également d'assurer la protection de la vie privée du conducteur et des passagers. En outre, un système de détection d'intrusion classique intégré dans le cadre de la sécurité routière est présenté. Finalement, une autre approche qui est à même de garantir des performances de faible ECUs est introduite.

Dans le chapitre 6, *Prototypes et leur Évaluation*, nous présentons des simulations et des mesures de la mise en œuvre de ces techniques et discutons les décisions de conception et les détails du prototype.

La thèse se conclut par la présentation de *conclusions et perspectives* dans le chapitre 7. Nous discutons de la faisabilité de nos approches, de l'applicabilité des techniques et de leurs avantages et inconvénients respectifs. Nous donnons également un aperçu des travaux futurs sur les sujets abordés dans la thèse, ainsi que des perspectives en matière de recherche et de développement industriel.

Dans l'annexe B, nous fournissons des *Mesures supplémentaires et détails de mise en œuvre*. Dans l'annexe C.1 (page 181) les *Sigles et acronymes* qui sont utilisés dans la thèse sont développés et expliqués. La version électronique du document permet de naviguer sur cette page en cliquant sur une abréviation.

A la fin de la thèse, des listes de *Figures, Inscriptions et Tableaux* sont fournies. Une liste des publications effectuées lors des travaux de cette thèse est donnée par la suite. Une *bibliographie* qui donne la liste des références aux travaux cités complète enfin cette thèse.

A.2 État de l'Art

Nous donnons dans cette section un aperçu de l'état de l'art de la sécurité réseau et logicielle des systèmes embarqués véhiculaires. Nous montrons également les développements en cours et à venir dans les mécanismes de sécurité pour véhicules et de l'électronique embarquée et des réseaux.

A.2.1 Les Systèmes Véhiculaires Actuels

Les véhicules ont longtemps été un domaine purement mécanique. Cet état de fait a radicalement changé ces dernières décennies. À compter de la gestion électronique du moteur dans les années 70, les véhicules ont évolué vers une plateforme informatique multi-connectée. Dans le même temps, les systèmes de sécurité routière autrefois mécaniques ont évolué vers des équipements électroniques (ESP, ABS) qui ont été introduits avec succès. Des systèmes de communication véhicule-à-véhicule embarqués sont en train d'être déployés et constituent un premier pas vers la conduite autonome. Le déploiement commercial d'un système conduisant de manière autonome est encore irréaliste dans les années qui viennent, mais les défis récents de la DARPA pour la conduite autonome ont remarquablement montré sa faisabilité. Entre 2010 et 2012, des tests sur route ont été réalisés à grande échelle avec plusieurs centaines de véhicules. Ceux-ci ont conduit à la délivrance d'une autorisation officielle pour l'usage routier des véhicules autonomes dans deux états des États-Unis en 2012.

Électronique Embarquée Capteurs et appareils de contrôle sont devenus de plus en plus complexes ces dernières années. Ceci est également dû au fait que la redondance joue un rôle important pour la réduction des risques en ce qui concerne la sécurité routière. Un exemple concerne le placement des capteurs redondants utilisés pour le freinage électronique ('brake-by-wire') : il n'y a pas un seul capteur à la pédale, mais un ensemble de plusieurs capteurs redondants, qui mesurent différentes grandeurs physiques. L'un des capteurs est utilisé pour la mesure de l'inclinaison de la pédale de frein et un autre pour détecter la pression appliquée à la pédale. Ces capteurs sont généralement intégrés dans un boîtier de capteurs connecté aux bus du véhicule. Un exemple d'implantation d'un capteur de pédale de frein électronique est décrit dans [ISS02] à partir de 2002. Nous l'avons schématiquement illustré à la figure 2.2 page 12. Alors que la pédale d'accélérateur est électronique dans la plupart des véhicules vendus aujourd'hui, les systèmes de freinage électronique sans solutions de secours mécanique ne répondent pas encore aux exigences légales dans tous les marchés. Une faille dans le logiciel de l'unité de freinage a été la cause d'un rappel important de véhicules par Toyota en 2010 [Wil10]. En Octobre 2012, Nissan a annoncé mettre en place un système de direction électronique (direction-by-wire) dans la production de

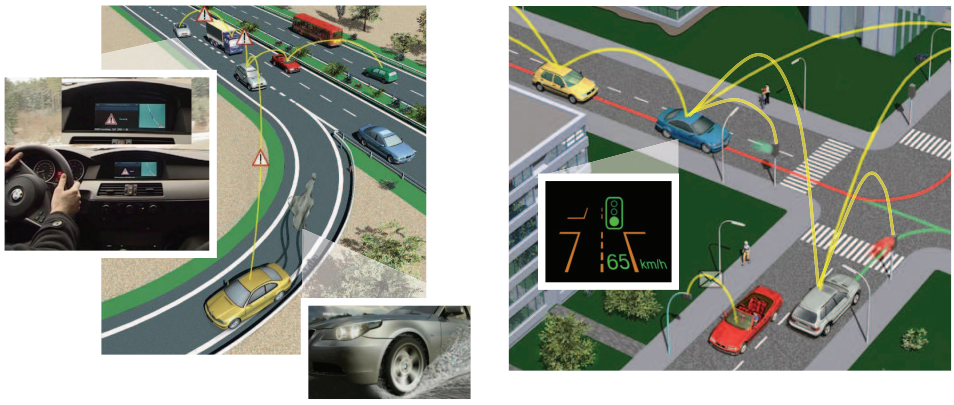


Figure A.2: Exemples de communication Car2X : Messages DENM sont émis et transmis à avertir les autres conducteurs (à gauche) ou pour améliorer l'efficacité du trafic, par exemple les horaires de diffusion des feux de circulation (à droite). Photos utilisées avec l'autorisation de BMW Forschung und Technik GmbH.

série pour les véhicules Infiniti dans moins d'un an [Nis12]. Leur système utilise une configuration redondante de trois calculateurs et un système de sauvegarde mécanique pour contrôler électroniquement le système de direction du véhicule, et donner une rétroaction au volant de manière sélective. La colonne de direction mécanique peut remplacer des signaux électroniques à l'aide d'un embrayage, qui découple le volant des roues motrices en mode de fonctionnement normal. Depuis 2005, un tel système d'urgence redondant ne serait même pas nécessaire selon une révision du règlement de la ECE qui englobe notamment le pilotage électronique ('steer-by-wire') [Uni05].

VANETS : La communication véhicule-à-véhicule Malgré le fait que la communication Car2X n'est pas encore déployée dans les véhicules d'aujourd'hui, les résultats préliminaires de nombreux projets de recherche et de tests sur le terrain commencent à être disponibles [FFH+04, PBH+08, SKL+06]. L'objectif de cette thèse est la communication à bord, mais celle-ci joue également un rôle très important dans la confiance mise en place de systèmes Car2X. Actuellement, une approche basée sur des certificats avec différents niveaux de confiance est envisagée pour représenter la sécurité vis-à-vis des communications internes au véhicule.

Dans les systèmes Car2X, il existe principalement deux types de messages. Les messages appelés balises Cooperative Awareness Messages (CAMs) sont diffusés régulièrement pour annoncer la position du véhicule, sa vitesse et son accélération. Le deuxième type de messages est constitué des messages Distributed Environment Notification Messages (DENMs), qui servent pour avertir d'un danger et des conditions routières. Alors qu'un CAMs n'est pas relayé, les messages DENM peuvent être transmis par des noeuds. Ceci est utile, par

exemple, pour transmettre un message d'alerte en amont le long d'une route avec des voitures circulant en sens inverse.

En revanche, les messages des balises Cooperative Awareness Messages (CAMs) sont utilisés pour éviter les collisions : une prévision de trajectoire est calculée pour les véhicules environnants, et par rapport à la trajectoire propre du véhicule, de sorte que le conducteur peut être averti en conséquence.

Par contre, les DENM sont envoyés, par exemple, dans les situations d'urgence : si un accident a déjà été détecté, ou si un freinage d'urgence est effectuée. Le relayage de ce type de message permet de s'assurer non seulement que les véhicules dans le voisinage, mais aussi la circulation en amont recevront cette information. Ces messages sont également utilisés pour diffuser l'information routière, tels que les fréquences des feux de circulation.

Un facteur important dans la communication sans fil est la sécurisation appropriée du contenu : comme mentionné ci-dessus, la cryptographie à courbes elliptiques (ECC) est prévue pour des signatures cryptographiques Car2X de communication, en raison de la longueur plus courte des clés et des signatures. Des certificats de pseudonymes sont utilisés pour signer les données envoyées : l'identité du véhicule ne peut pas être facilement révélée. Cela permet de préserver la vie privée des conducteurs et a été évalué dans le projet européen dédié PRESERVE [PRE14]. Chaque véhicule est titulaire d'un certificat d'identité à long terme. Cette identité est utilisée pour l'authentification par une infrastructure de clé publique (PKI) centralisée afin de récupérer des certificats de pseudonyme frais, qui sont ensuite utilisés sur la route.

L'état actuel de la recherche dans les réseaux véhiculaires est décrit dans [HL10]. Le livre couvre toutes les couches de communication et comprend un chapitre sur les approches securité dans le Car2X et les nouvelles normes.

Dans un futur proche, des services de sécurité routière et des applications seront déployées dans les véhicules ainsi que dans l'infrastructure routière. Cette possibilité est à l'étude dans le Consortium Car2Car et l'organisme de standardisation ETSI.

Afin de fournir une base fiable pour la communication inter-véhicules, sécuriser les communications à bord des véhicules est crucial.

Dans les premières générations de Car2X, les véhicules utiliseront les informations fournies par les RSUs et sont conscient des autres véhicules dans le voisinage, ce qui leur permet d'avertir le conducteur des dangers à venir tôt et de manière adéquate. Alors que les solutions de sécurité généralement propriétaires ont généralement été appliquée pour les fonctions du véhicule classiques, tels que les odomètres numériques et le contrôle d'accès physique (par exemple, porte-clés pour ouvrir les portes à l'aide de communication radio sans fil), inter-véhicule de communication doit s'appuyer sur des normes bien définies qui ont d'être reconnu et respecté par tous les fabricants et fournisseurs.

Ces nouvelles applications offrent de nouvelles surfaces d'attaque, qui peuvent même être exploitées à distance et peuvent entraîner des conséquences graves. Sans mesure de sécurité supplémentaire, de fausses alertes peuvent être créées et des attaquants peuvent exploiter des applications de sécurité en leur faveur, mettant ainsi en danger la sécurité des conducteurs. Nous décrivons dans ce manuscrit des exemples d'attaques réussies pour illustrer le besoin de sécurité.

Sécurité des VANETs Après que les premièrs systèmes coopératifs de communication de véhicule à véhicule ont été testés avec succès au début des années 2000 [FFH+04], le besoin de sécurité est devenu évident, car des attaques réussies sur les fonctions de sécurité peuvent non seulement perturber le système, mais aussi conduire à des accidents mortels. Le projet SeVeCOM [SeV08] a évalué ce besoin de sécurité et a mené plusieurs expériences en 2006. Il s'est avéré que les certificats, qui sont nécessaires pour pour authentifier les véhicules à distance, peuvent causer une surcharge importante en communications, ce qui a entraîné le choix d'algorithmes cryptographiques à base de courbes elliptiques (ECC). Leurs clés et la longueur de leurs signatures sont courts, mais offrent le même niveau de sécurité que RSA avec des clés plus longues [LBH+06]. Des optimisations de l'utilisation des canaux, par exemple en omettant intelligemment et autant que possible des certificats [Pap09] ont été développés dans ce projet. Le réseau d'excellence en systèmes informatiques de sécurité, SysSec, a publié un rapport sur l'état de l'art de la sécurité dans les véhicules connectés [Sys12]. Il couvre le Car2X ainsi que la sécurité interne des véhicules.

Systèmes Coopératifs Routiers Expérimentaux Les tests opérationnels de systèmes coopératifs (Field Operational Tests (FOTs)s) sont des programmes qui permettent d'évaluer la faisabilité et les performances des communications inter-véhiculaires par des bancs d'essais réalistes. Dans ces FOTs, un certain nombre de véhicules, en général entre 10 et 200, sont équipés de systèmes embarqués et sont exploités dans une zone spécifique qui est équipée avec des RSUss. La sécurité cryptographique pour la communication sans fil est appliquée dans tous les FOTss actuels. Par exemple, le projet sim^{TD} utilise une infrastructure à clé publique à long terme ainsi que des identités de courte durée, ou pseudonymes. Celles-ci sont établies et exploitées [BSM+09] à travers une acPKI qui a été mise en place par le consortium du projet. Le concept d'identité à court terme est lié à la procédure d'attestation anonyme directe, que l'on trouve dans les architectures de TPM. En raison de la puissance de traitement limitée, les FOTs survolent généralement l'implantation de la partie cryptographique des protocoles mis en Ûuvre. Par exemple, au lieu de cryptographie à courbe elliptique, qui est prévu pour la normalisation, RSA-512 est utilisé pour les certificats à court terme dans sim^{TD}.

Des tests similaires existent dans d'autres pays, comme par exemple SCORE@F

en France. Ces FOTs sont coordonnés par le projet routier C2X, qui effectue des tests d'interopérabilité dans le cadre de la normalisation de l'ETSI.

Le résultat des FOTss va largement influencer la normalisation et la faisabilité industrielle. Aux États-Unis, les projets VII et VSC ont déjà abouti à la norme IEEE WAVE [HL10, US 09]. L'état actuel de la normalisation et des défis pour les États-Unis est présenté dans [LLZ+08]. La Commission européenne a donné un mandat de normalisation à l'ETSI et au CEN [Eur09b], qui couvre la normalisation des communications et applications automobiles. Une forme modifiée des spécifications WAVE (IEEE 1602.11) est utilisée pour les FOTss en Europe. En dehors des systèmes véhicule-à-véhicule, le système d'urgence eCall [Eur09a] va être obligatoire pour les véhicules neufs dans l'Union européenne. Nous allons voir de plus en plus l'intégration de matériels et logiciels de communication dans les véhicules dans les prochaines années, ce qui exige une sensibilisation accrue à la sécurité.

Sécurité des réseaux embarqués La nécessité de la protection des données à l'intérieur des véhicules est mentionnée dans plusieurs publications et projets de recherche [Kun08, EB06, GFL+07]. Néanmoins, aucun de ces projets n'a étudié les solutions possibles pour la sécurité dans une plate-forme automobile et leurs implications. Le projet européen EVITA vise exactement cet objectif d'assurer la sécurité et la confiance à l'intérieur du véhicule. Il est déjà nécessaire d'ajouter des mesures de sécurité au niveau des capteurs, parce que dans les données en provenance de la voiture de données peuvent activer des applications impliquant la communication véhicule-à-véhicule. Ainsi, des messages internes, tels que le déploient les airbags ou le freinage d'urgence, peuvent entraîner la diffusion de messages d'avertissement à l'extérieur du véhicule.

Des mesures de sécurité ont jusqu'à présent été appliquées pour des fonctionnalités très spécifiques et dans un peu de composants du véhicule. Par exemple, le téléchargement des mises à jour du micrologiciel des calculateurs embarqués est protégé par un code d'accès de deux ou plusieurs octets. Bien sûr, un code à deux octets ne fournit qu'une sécurité marginale contre des attaquants avec un temps et un nombre d'essais illimités. [KCR+10] a démontré que le code de sécurité requis par la norme peut être trouvé en seulement trois jours et demi en effectuant une attaque par force brute (essais et erreurs). L'introduction de signatures cryptographiques fortes a d'ailleurs été récemment décidée par le groupe de fabricants HIS en Allemagne [ZS08].

Dans le cadre du projet EVITA, le but était d'associer des systèmes de sécurité pour les réseaux intra-véhiculaires avec la communication externe et les applications de sécurité correspondantes, ainsi que de fournir une base pour de futurs projets concernés par la sécurité dans le véhicule. Le projet européen OVERSEE met l'accent sur les environnements d'exécution sécurisés dans les véhicules [GHR+09] et le projet SEIS allemand va mettre en œuvre la sécurité du

protocole Internet (IP) dans les architectures de réseau de bord de l'industrie automobile [GHM+10]. Récemment, quelques nouvelles approches pour protéger la sécurité routière CAN ont été proposées [CRH05, NLJ08, OYN+08, GR09, Be09, HSV11]. Le bus FlexRay, qui est censé succéder à CAN, présente des lacunes de conception très similaires en ce qui concerne la sécurité, et notamment l'écoute et l'injection de charge utile sont encore possibles [NLPJ09]. Certaines approches nécessitent de déployer des circuits électroniques d'émission-réception adaptés et différentes technologies réseau. Toutes ces méthodes utilisent la sécurité cryptographique pour authentifier la charge utile du bus. Nous allons expliquer ces approches plus en détail dans la section 4.3. Dans l'ensemble, la sécurité des composants embarqués dans les véhicules a pris de l'intérêt pour les chercheurs comme pour l'industrie. Le fait que les sociétés de conseil de sécurité et antivirus publient enquêtes et livres blancs sur ce sujet [Lin09, MWW+11] illustrent également cet intérêt croissant. Une bonne vue d'ensemble sur les activités de recherche actuelles en matière de sécurité dans les véhicules est donnée dans l'état de l'art publié dans [KOJ11]. Le rapport SysSec détaille aussi l'état de l'art de la sécurité dans le véhicule connecté [Sys12].

Paradigmes de Communication Interne Actuels Le processus de développement utilisé à l'heure actuelle pour les applications réparties sur plusieurs ECUss s'explique par l'historique des technologies employées. Lorsque le bus CAN a été introduit, il a remplacé des fils électriques sous tension dédiés ou des signaux codés par un bus informatique, où l'information numérisée est transmise. Ainsi, les données sur le bus sont souvent appelées $signaux$. Ces signaux sont généralement transmis d'une manière périodique (avec des cycles typiques de 10 à 100 ms), ce qui permet des contrôles en boucle fermée.

Au cours du développement, ces signaux sont définis dans les outils logiciels qui sont disponibles à partir de différents fournisseurs (la suite Vector est l'un des plus couramment utilisés, RTaW-Sim en étant une alternative libre). Le développeur ajoute des signaux et leur représentation discrétisée correspondante. Ces signaux sont ensuite associés à un calculateur émetteur et un ou plusieurs calculateurs récepteurs sont sélectionnées. La matrice résultante est appelée $K\text{-}Matrix$. Elle décrit la représentation des données, et la relation émetteur / groupe de récepteurs. Les ECUss avec plusieurs interfaces réseau peuvent assurer la propagation de trames et constituent alors des $passerelles$ au sein du réseau.

Les données modélisées peuvent être exportées dans des formats spécifiques décrivant le bus de communication. Un format basé sur XML appelé FIBEX (Format Field Exchange Bus) est en train de devenir un standard industriel. Il a été défini par l'organisme de normalisation ASAM (Association de normalisation des systèmes automatiques et de mesure).

Jusqu'à présent, les modèles et les outils de développement ne comprennent pas les moyens d'accroître la charge utile des données authentifiées et chiffrées. Bien

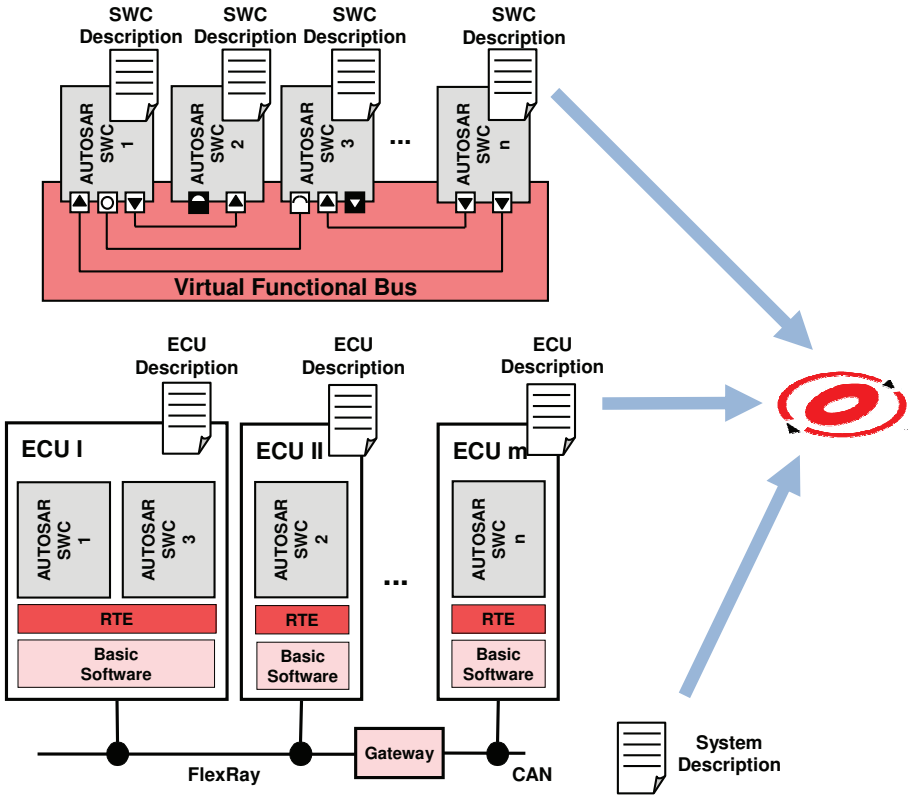


Figure A.3: AUTOSAR développement de paradigme : les composants logiciels (SWC) d'échanger des informations via le bus de fonction virtuelle (VFB). Ces fonctions logicielles sont déployées sur des calculateurs différents. Les modèles sont décrites dans un format lisible par une machine et compilées dans l'architecture globale du véhicule pour le déploiement. Le cadre d'exécution est partiellement généré à partir de la "configuration d'AUTOSAR", qui comprend tous les fichiers de description. Illustration de [BFWS10].

que les informations concernant la composition des groupes de communication, la fréquence des messages qui leurs sont destinés et leur taille sont explicitement disponibles, elles ne sont pas encore utilisées pour générer ou définir des politiques de contrôle d'accès au réseau ou pour détecter des données d'origine malveillante et non conformes.

Un Nouveau Paradigme : AUTOSAR L'architecture AUTOSAR est basée sur un paradigme qui a été promu pour une utilisation dans les réseaux de véhicules depuis le milieu des années 2000. Il se compose d'un système d'exploitation embarqué (le système AUTOSAR, basé sur OSEK, un système d'exploitation automobile normalisé ISO [ISO05]) et d'une vue abstraite sur toutes

les communications, c'est-à-dire, le concept abstrait d'un bus de fonctions virtuel (VFB), qui remplace la communication explicite entre processus et le réseau d'échange de données. Cette approche d'ingénierie des modèles permet de décrire le placement des composants logiciels (SWC) sur les différents calculateurs distribués à un niveau abstrait de compilation et de déploiement "presse-bouton". Le système d'exploitation fournit un cadre pour les fonctions du logiciel et des tâches qui peuvent accéder à du matériel et des bibliothèques à travers une interface de programmation standardisée. La figure A.3 montre le paradigme de développement, avec une compilation de tous les modèles dans une architecture de déploiement.

Depuis sa dernière version majeure, AUTOSAR 4.0 dispose d'une bibliothèque de services cryptographiques qui fait partie de sa spécification [BFWS10]. Cette bibliothèque fournit des fonctions pour les applications et des fonctions du système d'exploitation en ce qui concerne des algorithmes spécifiques : MD5, SHA-1, RSA, AES et d'autres peuvent être accessibles via la bibliothèque CryptoAbstractionLibrary [AUT11].

Malgré la standardisation de l'API Crypto, il n'existe pas encore de moyen de sécuriser les communications entre des liens de communication AUTOSAR. En particulier, si une communication sécurisée doit être mise en œuvre, la charge utile de sécurité devrait être ajoutée à la couche application. C'est ce qui se fait actuellement pour les fonctions système telles que la mise à jour du firmware ou certaines fonctions de diagnostic.

A.2.2 Techniques pour Sécuriser l'Exécution de Logiciels

Suivi dynamique de la circulation de l'information La technique DIFT permet de s'assurer que les données en entrée (par exemple, les données entrées par l'utilisateur du réseau, le fichier d'entrée,) peuvent être étiquetées en interne ($contaminé$). Cette balise est effectué tout au long de l'exécution. Ces balises peuvent représenter une donnée en entrée d'origine inconnue ou non fiable. Les applications téléchargées peuvent aussi utilement être marquées. Au moment de l'exécution, les données en entrée de l'application sont automatiquement marquées. Lorsque l'application traite des données marquées, les données lues à partir de ces descripteurs de fichiers héritent automatiquement du marquage, c'est à dire que la balise se propage le long de la chaîne suivant laquelle circule l'information au cours de l'exécution d'une application. Ces fonctions de propagation peuvent être décrites dans un cadre logiciel comme par exemple [LC06a]. Un tableau global de zones de mémoire marquées, souvent appelée mémoire shadow, est conservé par l'environnement d'exécution et évalué lorsque la mémoire est accédée. Pour toutes les opérations d'affectation, la nouvelle information de marquage (généralement un vecteur de bits) pour l'opérande gauche est dérivé des informations de marquage de l'opérande de droite à l'aide de la fonction

d'héritage ou de propagation mentionnée ci-dessus. Policy Scope [ZJS+09] est par exemple un outil qui suit cette approche.

> ... "Le suivi de la circulation de l'information (IFT pour *Information Flow Tracking*) se réfère à la possibilité de suivre la façon dont le résultat de l'exécution d'un programme est marqué, par l'intermédiaire soit des données ou des dépendances de contrôle, à ses entrées à partir du réseau, du système de fichiers et des autres paramètres externes tels que les variables d'environnement et les arguments de ligne de commande. Pour suivre avec précision les flux d'information, chaque donnée en entrée doit être marquée par une étiquette, qui peut être un bit (par exemple, un bit 'taint' ou marquage) ou un pointeur vers une méta-structure de données arbitrairement complexe, et pour chaque opération d'affectation, l'étiquette de la partie gauche de l'opération est dérivée de celle de sa partie droite selon des règles de combinaison des marquages. Différentes applications de suivi des flux d'information exigent différents types de marquages et d'utiliser différentes règles de combinaisons de balises de marquage" ...
>
> Lam and Chiueh in [LC06a].

Une solution complémentaire serait de suivre $comment$ les données sont utilisées au niveau applicatif, tout spécialement dans les environnements mobiles où la nécessité d'un contrôle d'accès pour les applications (par exemple, l'accès à des données privées telles que l'annuaire et des informations de localisation) a été reconnue. Le système TaintDroid vise exactement cette application [EGC+10]. TaintDroid est une extension du système d'exploitation Android (lui-même basé sur Linux) et de son environnement d'exécution (DEX ou Dalvik Execution Environment, une implantatation de Java). Il définit plusieurs sources de marquage (taint), qui contiennent des données relatives à la vie privée, et suit l'information tout au long de la durée de vie d'une application. Il utilise des règles logiques pour propager des balises de marquage dans la mémoire et lors de l'exécution. En considérant le code provenant de bibliothèques comme de confiance, TaintDroid parvient à obtenir un faible surcoût d'exécution, en moyenne de 14% plus élevé. La figure A.4 montre la propagation de données de marquage dans la mémoire.

Lorsque le code binaire n'est pas interprété mais directement exécuté, l'instrumentation binaire peut être utilisée pour injecter des fonctionnalités de marquage à l'exécution. Nous discutons des approches existantes dans le chapitre correspondant sur la page 94.

Contrairement à l'instrumentation du code interprété ou binaire, il existe aussi des outils de traduction source-à-source, qui produisent du code C traçable. Les

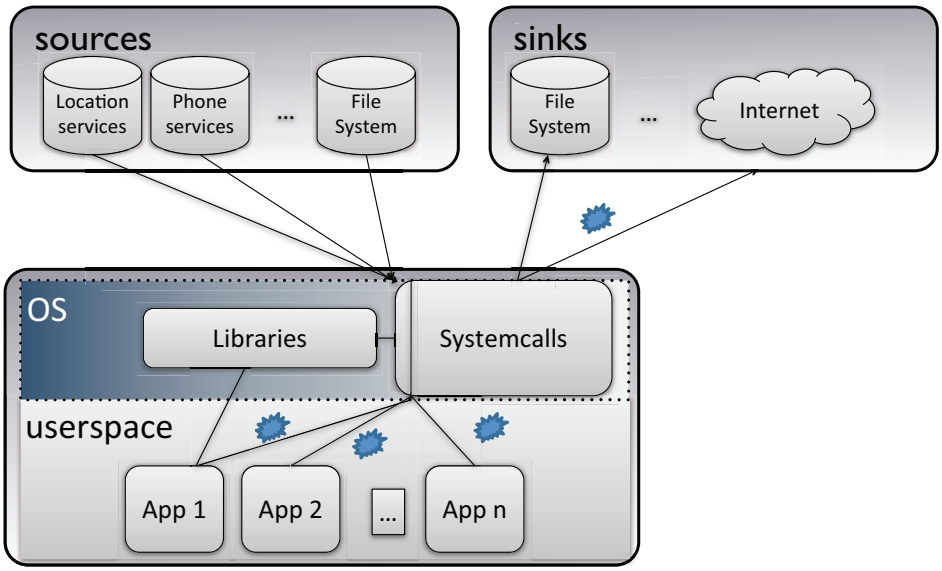


Figure A.4: Propagation des données marquées (bleu) dans la mémoire telle que transposée dans TaintDroid. Les données provenant de sources de marquage sont marqué daes que fournies par les services du système d'exploitation (appels systèmes, accès au système de fichier). Les bibliothèques sont considérées comme dignes de confiance et ne sont donc pas marquées par le moteur d'IFT [EGC+10].

cadres logiciels GIFT [LC06b] et DIVA [XBS06] appliquent cette technique. Bien que cette technique a l'avantage d'un surcoût limité, son inconvénient majeur est que le code source des programmes doit être disponible. En tant que tel, le suivi de flux d'information source-à-source ne peut pas être appliqué à des programmes binaires propriétaires.

Une approche basée sur le matériel montre que le traitement des marques n'a pas besoin d'être limité à un code interprété ou exécuté symboliquement. Dans [KDK09] un co-processeur permet de suivre les données marquées de programmes génériques s'exécutant sur le processeur hôte.

Surveillance de la circulation des informations dans les environnements distribués Un cadre logiciel pour la surveillance de flux d'informations dans les environnements distribués sous le nom de Aeolus a été proposé dans [Che09]. Son modèle de sécurité est basé sur le contrôle de flux d'informations et introduit un certain nombre de mécanismes qui sont appliquées au code qui est exécuté à un niveau intermédiaire (par exemple, Java et .NET). De même, un système appelé Laminar met également en place sur les machines virtuelles Java et y introduit des étiquettes [RPB+09]. Ces étiquettes concernet soit la sécurité soit des autorisations. Le concept d'étiquette est basé sur la classification des informations, telles que celles utilisées par les services militaires de recherche d'infor-

mation. Ces cadres visent la confidentialité de certains renseignements et documents. Leur approche utilise des fonctionnalités d'intergiciels, et ils ne suivent donc pas les flux de données indirects à l'intérieur de l'application, comme cela peut être fait avec les approches d'instrumentation binaire. Une approche qui utilise une instrumentation binaire locale avec des connexions TCP/IP instrumentées pour propager l'étiquette avecdes "gestionnaires de circulation" (*Flow Managers*) est présentée dans [KKCS09]. Plus récemment, libdft [KPJK12] fournit une boîte à outils pour le suivi des flux d'information, qui a été utilisée pour distribuer des étiquettes entre des processus et entre des noeuds [ZPK11].

A.2.3 Conclusion

La sécurité fait partie intégrante des systèmes informatiques d'aujourd'hui. Malgré le fait que les programmes antivirus et autres solutions de détection d'intrusion fournissent une protection "visible" et souvent efficace, l'exigence principale de sécurité est la définition de principes architecturaux solides. Cela est vrai pour n'importe quel type de système d'exploitation, d'environnement d'exécution, ou d'infrastructure réseau.

Ordinateurs et réseaux n'étaient pas bien protégés dans les premiers jours de l'Internet, comme l'a démontré en 1988 Robert Morris Jr. avec son ver [Orm03], ou [And02] pour les ordinateurs personnels qui n'avaient pas de liens de communication. Ce paradigme est maintenant bien compris et la sécurité partie intégrante de tous les principaux systèmes d'exploitation.

Nous assistons à l'ouverture du domaine véhiculaire à l'intégration de logiciels et appareils ou dispositifs tiers, créant la nécessité de protéger les systèmes contre des attaques nouvelles. Bien qu'il soit important de protéger ces systèmes, la sécurité est aujourd'hui rarement prise en compte dans les processus de spécification et de test des calculateurs embarqués et de l'architecture de communication.

Des approches existent pour sécuriser des éléments spécifiques, tels que les dispositifs d'immobilisation de véhicules. Toutefois, un concept global de sécurité pour les réseaux embarqués et les unités de contrôle des véhicules couvrant l'infrastructure véhiculaire complète n'a pas encore été déployé. Plusieurs domaines prennent actuellement de l'importance dans l'environnement du véhicule, ce qui rend l'application combinée des concepts de sécurité difficile, car ils reposent sur des techniques différentes et des mesures qui sont parfois incompatibles. Une solution globale à ce problème est toujours manquante.

Etat de la Sécurité des Réseaux Embarqués La sécurité de l'électronique dans l'architecture des véhicules a jusqu'ici principalement été maintenue par l'isolement physique et le secret de mise en œuvre. Le réseau d'un véhicule

n'est pas facilement accessible car il constitue fondamentalement un système autonome avec des capteurs qui le connectent au monde extérieur. La connexion physique aux cables d'un bus est nécessaire pour interagir avec les composants du réseau, c'est à dire les calculateurs. Le principe de "l'isolement et l'obscurité" a été utilisé depuis plusieurs décennies dans l'industrie automobile, par exemple pour les dispositifs d'immobilisation tels que mentionnés ci-dessus et ont fait long feu. Bien qu'ils semblent bénéfiques à première vue, ils ne peuvent pas remplacer une architecture de sécurité correcte sur le long terme. L'approche mise en œuvre pour garder les détails secrets constitue une violation évidente du 2ème principe de Kerckhoffs d'*absence de sécurité par l'obscurité* [1].

L'obscurité de la conception des protocoles pour le bus CAN et de leurs identifiants, par exemple, n'existe plus aujourd'hui. Les circuits électroniques d'émission-réception sont librement accessibles et de nombreux identifiants CAN, encore aujourd'hui confidentiels pour les industriels, sont largement publiés sur l'Internet, ou peuvent être identifiés par ingénierie inverse. Ainsi, seul l'emplacement du câble (donc l'accès physique) reste secret, un secret sur lequel repose toute la sécurité d'un véhicule. Plus encore, de nombreux appareils qui sont connectés au réseau interne exposent, dans le même temps, une interface accessible par l'utilisateur qui peut même être compromise à distance dans certaines circonstances.

Nous pensons qu'une adaptation du modèle de développement est nécessaire et doit inclure la sécurité dès la conception. Ce changement de paradigme a été récemment mis en branle dans l'industrie par la publication d'analyses de sécurité sur les véhicules connectés.

[1] 2ème principe de Kerckhoffs : le système ne doit pas reposer sur un secret, tel qu'un secret perdu brisera le système. («Il faut qu'il n'exige pas le secret, et qu'il puisse sans inconvénient tomber entre les mains de l'ennemi»), [Ker83].

A.3 L'Environnement et les Composants

Modèle d'Attaquant De nombreuses motivations existent pour les attaquants des véhicules modernes. La première consiste à s'introduire dans un véhicule pour le voler. Bien que ce soit certainement une menace réaliste pour la sécurité du véhicule, notre modèle prévoit des attaques plus sophistiquées contre le véhicule. De telles attaques peuvent provenir du propriétaire du véhicule lui-même, notamment dans le but d'ajuster et régler la fonctionnalité du véhicule, tels que :

- Modification des paramètres du moteur pour des performances accrues,
- Falsification du kilométrage du véhicule,
- Suppression de restrictions techniques (par exemple, regarder la télévision pendant que la voiture est en mouvement),
- Déblocage de fonctionnalités et logiciels (par exemple, système de navigation),
- Ajout de fonctionnalités (par exemple, la compilation et l'exécution d'applications propres, en changeant le système de navigation).

Bien que certaines mesures, comme la modification du système d'autoradio ou de navigation peuvent être mineures, d'autres influent sur la sécurité des véhicules (regarder la télévision pendant la conduite) ou génèrent une perte financière ou des fraudes (déverrouillage des fonctions rémunérées, changement du kilométrage).

Comme un véhicule est un système sensible aux modifications, les fabricants rejettent habituellement toute responsabilité en cas d'altération du système fermé mis au point par l'industriel (par exemple, un dispositif supplémentaire est ajouté au réseau du véhicule, ce qui peut perturber le fonctionnement des systèmes de sécurité routière). En raison de l'énorme demande de contenus dynamiques et les attentes toujours croissantes des conducteurs, les fabricants sont obligés d'autoriser des contenus en ligne modifiables en étroite interaction avec leurs systèmes embarqués de divertissement. Le véhicule expose plusieurs interfaces, allant de l'appairage des équipements USB ou Bluetooth, en passant par l'accès à des serveurs back-end, jusqu'à des connexions mobiles 3G ou LTE. Cela crée une surface d'attaque supplémentaire par rapport à l'existant, qui fonctionne en circuit fermé.

Pour nos scénarios, nous ne différencions pas entre les attaquants internes et externes, mais supposons que l'attaquant peut avoir un accès physique au véhicule, au moins pour une courte période. Cela signifie que l'attaquant peut lire des données à partir de réseaux internes et injecter ou relire les données. L'attaque elle-même peut avoir lieu à un stade ultérieur, par exemple, en utilisant les

données obtenues ou injectées, ou par un code à distance. Une attaque à distance offre des possibilités comparables si l'attaquant peut accéder librement aux interfaces réseau, après avoir compromis un calculateur (ECU) avec sucès.

Definition: Le *modèle d'attaquant* est défini par les capacités et contraintes suivantes. Un attaquant peut accéder à toutes les interfaces de communication du véhicule : il peut lire les données sur les bus et sur les interfaces, et aussi injecter des données. L'attaquant peut télécharger ou remplacer certains programmes des ECUss du véhicule, mais ne peut pas accéder aux données internes du HSMs qui fait partie de l'ECU. Nous supposons que le processeur combiné avec le HSM est inviolable et aucune attaque invasive, tels que micro-sondage ou des attaques physiques (par exemple, dans le but de mettre un zéro la mémoire ou de modifier les fusibles du microcontrôleur) ne sont réalisés. Des leçons au sujet de ces attaques ont été tirées dans l'industrie de la carte à puce [KK99] et s'appliquent à ce domaine de façon similaire.

Notre définition couvre les attaquants internes avec un temps illimité, comme le propriétaire du véhicule, ainsi que les attaquants externes qui peuvent gagner un accès à court terme au véhicule (grâce à l'ingénierie sociale, la force physique ou des défauts de mise en œuvre au niveau des interfaces exposées, telles que les réseaux sans fil).

A.3.1 Module de Protection Matériel

Un aspect clé de l'architecture EVITA est le module de protection (ou de sécurité) matériel, le Hardware Security Module. Il stocke les matériels cryptographiques comme les clés et traite toutes les opérations cryptographiques, telles que les informations d'identification qui doivent toujours être gardées confidentielles, *ne sortira jamais de ce module en clair* et même ne sont jamais chargées dans les registres du CPU ou de la mémoire d'application. Seules quelques-unes de ces clés cryptographiques peuvent être obtenues par l'application durant une session de HSM.

A.3.1.1 Usage des clés

L'utilisation des clés est commandée par des indicateurs ou drapeaux d'usage (*use-flags*). Ces indicateurs fournissent un accès de bas niveau et le contrôle de l'utilisation cryptographique. En particulier, ils lient l'utilisation de cette clé pour certaines configurations à certaines fonctions, notamment à une authentification distincte telle qu'un mot de passe. Ces conditions doivent être remplies pour pouvoir utiliser une clé avec le HSM.

Indicateur d'Amorçage Sécurisé L'indicateur le plus important est le drapeau d'amorçage (boot) sécurisé : il lie une clé au succès d'une procédure de démarrage du système, connu par les valeurs de hachage présentes dans les registres de configuration des ECU (ECR). Ces registres travaillent comme le registre PCR d'un TPM, à savoir que des mesures sont prises à chaque étape du processus de démarrage. La liaison d'une clé de mesure ECR permet de s'assurer que la clé n'est pas utilisée par un firmware remplacé, et seulement dans l'environnement prévu et correctement initialisé. Le processus de démarrage sécurisé a été démontré sur un microcontrôleur Infineon TC1797 avec le système d'exploitation AUTOSAR. Les clés qui font partie des protocoles que nous présentons dans le chapitre 4 sont liées à un amorçage securisé.

Indicateurs de fonctionnement spécifiques Une clé peut être limitée à certaines opérations : elle peut, par exemple, être utilisée pour vérifier les signatures cryptographiques, alors que l'utilisation de cette clé pour la génération des signatures n'est pas possible. Nous avons construit notre mécanisme de distribution de clés autour de cette fonction afin de permettre à faible coût en termes de performances d'utiliser de manière asymétrique des clés symétriques.

A.3.1.2 Interface de programmation du HSM

L'interface de programmation du HSM est particulièrement simple et générique tandis que les algorithmes peuvent être complexes et fondamentalement différents comme par exemple les signatures asymétriques par rapport aux codes d'authentification de messages. L'interface de programmation de cette fonctionnalité est conçue pour être flexible et générique. Elle comprend un certain nombre de fonctionnalités essentielles : génération de clé, importation et exportation de clés (en utilisant le chiffrement de transport) et d'opérations cryptographiques (par exemple le chiffrement et le déchiffrement, en plus de la génération d'une signature et de sa vérification).

A.3.1.3 Plate-forme d'experimentation HSM

Le HSM est mis en œuvre dans le cadre d'un FPGA Xilinx Virtex-5 FPGA. Les opérations cryptographiques sont réalisées à l'intérieur du FPGA. Toutes les opérations d'E/S, le stockage des clés, et le micrologiciel sont mis en œuvre dans le cadre d'un Linux embarqué (v2.6.34), qui utilise la base Virtex440-PowerPC et 256 Mo de RAM disponibles sur cette plate-forme. En tant que prototype, le HSM est lui-même relié à l'unité centrale via Ethernet. Une interface SPI intégrée existe aussi mais n'a été utilisée que dans le cadre d'une preuve de concept de l'intégration d'AUTOSAR. L'interface Ethernet/IP utilise les sockets standards de TCP afin d'échanger des commandes HSM qui sont codées dans

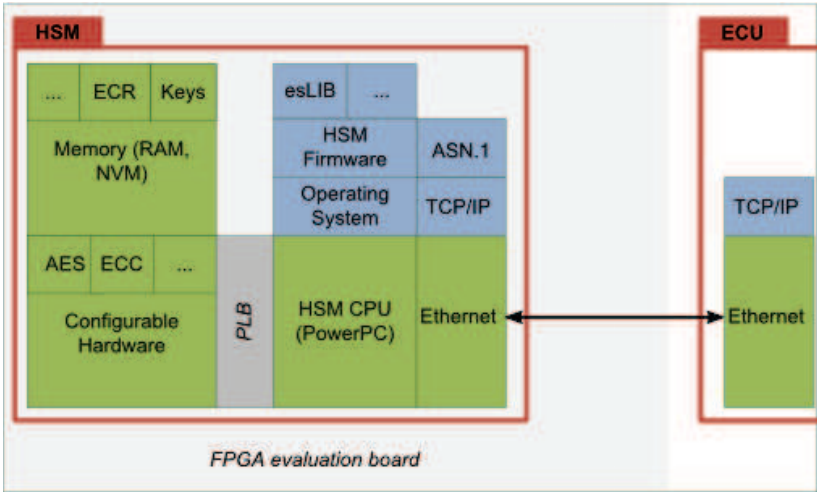


Figure A.5: HSM Prototype d'architecture matérielle : les fonctions cryptographiques sont déployées dans le matériel (vert) sur la carte FPGA. Le logiciel (en bleu), c'est à dire le contrôle de la fonctionnalité du HSM et sa disponibilité via une API de réseau (Ethernet, ASN.1). Dessin adapté de [PGW+11].

des paquets de requêtes / réponses ASN.1 L'architecture de la plate-forme prototype du HSM est décrite à la figure A.5.

A.3.2 Vers une Architecture Embarquée plus Sécurisée

Ce paragraphe traite de la problématique du matériel à bord des véhicules et de l'architecture des logiciels du projet européen FP7 EVITA. Cette thèse a été écrite en parallèle au travail réalisé dans le cadre du projet EVITA. Un certain nombre de contributions qui vont au-delà de ce qui est décrit dans la thèse, ont été apportées à la conception du protocole de l'architecture et à sa mise en œuvre.

L'architecture fournit une base pour des expériences sur les solutions de haut niveau développées dans cette thèse. Dans le cadre du projet EVITA, une analyse a été réalisée sur les scénarios d'attaque et les exigences de sécurité [RWW+09] concernant des cas d'utilisation typiques du monde de l'automobile connectée [KFL+09]. L'architecture de sécurité est une combinaison d'éléments matériels et logiciels [WWZ+10]. Le logiciel gère l'interface entre la sécurité et le fonctionnement des applications dans l'automobile, comme la communication authentifiée ou confidentielle. La partie matérielle fournit une base à la confiance : une plate-forme fiable pour le stockage de clé et des primitives cryptographiques. Cette co-conception matérielle / logicielle assure que les clés cryptographiques ne sont pas utilisées sans autorisation, à savoir, la plate-forme doit être correctement amorcée pour utiliser les clés, et seulement en fonction de leurs paramètres

d'usage. De plus les clés ne peuvent quitter le matériel dans une forme non chiffrée, c'est à dire que toutes les opérations cryptographiques ont lieu dans le module matériel. Remarque : La protection d'un HSM ne peut pas assurer l'intégrité des logiciels dans leur exécution. Ceci signifie que les attaques à l'exécution posent toujours un problème.

A.3.3 Scénarios

A.3.3.1 Scénario I : freinage actif (Car2X)

Un premier scénario concerne la communication Car2X. Nous avons décrit la nature générale des messages d'avertissement Car2X dans la section A.2.1, et nous employons dans ce scénario le concept de notification d'avertissement DENM comme exemple. Le scénario lui-même est basé sur deux véhicules distincts : un véhicule émetteur (SV) et un véhicule récepteur (RV). Le véhicule émetteur envoie le message d'avertissement DENM, qui est reçu par le véhicule récepteur. Ce message d'avertissement correspond à une situation de freinage d'urgence par le véhicule émetteur : un ESS est défini comme "*une force de ralentissement a été appliquée au véhicule*" [Uni10], les freins sont actionnés pour ralentir par $a \geq 6\frac{m}{s^2}$ [Uni11]. Le véhicule récepteur peut agir en conséquence, par exemple en affichant un message d'avertissement pour le conducteur et en se préparant à un freinage d'urgence (par exemple, en pré-chargeant les freins, en actionnant la ceinture de sécurité, et en fermant les fenêtres). En outre, si la confiance dans le contenu du message reçu est suffisamment élevée, les freins peuvent être actionnés de manière autonome. Les contrôles de plausibilité qui confirment la validité des données font partie de l'évaluation. Ces contrôles de plausibilité peuvent s'appuyer sur le calcul d'une trajectoire d'un véhicule en recueillant ses balises CAM ou sur le calcul d'une carte locale dynamique, comme prévu par la spécification de l'ETSI, ou avec le soutien des autres capteurs locaux tels que le radar, si la situation le permet.

Une vue abstraite du scénario est illustrée à la figure A.6, où les deux véhicules impliqués et leurs ECUs sont montrés.

Une vue détaillée et la séquence de messages échangés se trouve à la page 42, à la figure 3.3.

Dans les deux véhicules, le protocole de distribution de clés fait indirectement partie de la chaine de communication tel qu'expliqué dans la section suivante.

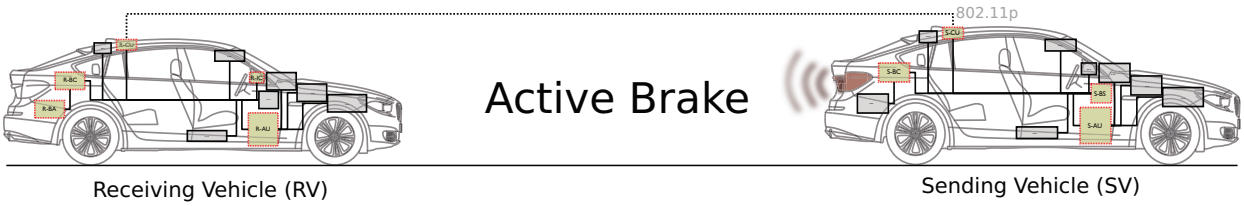


Figure A.6: Un freinage d'urgence est émis par SV (le véhicule émetteur) et reçu par RV (véhicule récepteur). Les ECUss participants sont marqués en jaune.

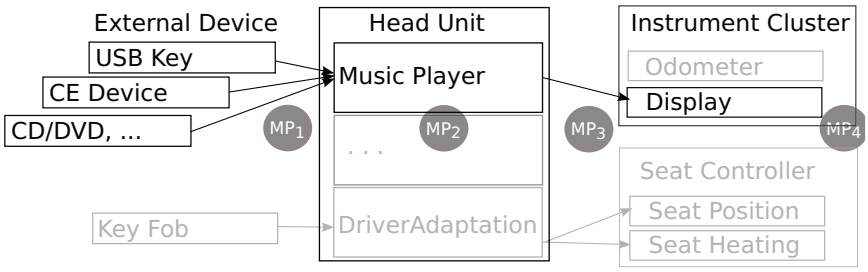


Figure A.7: Scénario II : Écoute de la musique sur l'unité principale. Les métadonnées de la musique sont extraites et envoyées au panneau d'instruments, où elles sont affichées pour le conducteur.

A.3.3.2 Scénario II : Diffusion Audio

Les passagers écoutent des enregistrements numériques dans le véhicule depuis que le disque compact a remplacé les cassettes. Cependant, les enregistrements numériques peuvent être lus de nos jours dans différents formats, par exemple MP3, WMA, M4A, et à partir de diverses sources comme les CD, clés USB, cartes SD ou sans fil depuis un téléphone mobile. Ces fichiers de différents formats offrent déjà une grande surface d'attaque qui peut être utilisée pour compromettre l'unité de divertissement, comme démontré avec succès à l'aide de fichiers WMA spécialement conçus à cet effet [CMK+11]. Des métadonnées comme le nom l'artiste ou le titre sont directement extraites des fichiers musicaux. L'analyse du fichier de métadonnées et de la musique ainsi que leur décodage sont effectués sur l'unité principale. Les métadonnées sont transmises au tableau de bord du véhicule, c'est à dire, l'écran placé à côté de l'indicateur de vitesse, où le titre de la chanson et l'artiste sont affichés au conducteur.

Ce scénario porte sur la lecture de musique à partir d'une source non fiable, par exemple, un fichier sur un disque, et la diffusion d'informations dans le réseau de bord. Une représentation du scénario est donnée à la figure A.7. Le diagramme des séquences de message se trouve sur la page 3.5 à la figure3.5.

A.3.3.3 Scénario III : Adaptation au conducteur

Les véhicules haut de gamme sont souvent équipés de jusqu'à une douzaine de moteurs à courant continu pour ajuster la position du siège. Ils offrent également la possibilité de conserver différentes positions du siège en mémoire. Certains véhicules combinent cette fonction avec l'identification d'un porte-clés spécifique. Dans notre troisième scénario, une application sur l'unité principale reçoit des informations depuis le porte-clés (par exemple, son identifiant) et ajuste la position du siège en fonction d'un profil enregistré (sur le disque). En réalité, cette étape devrait normalement impliquer au moins un composant intermédiaire, qui contrôle la communication radio avec la clé. En outre, afin

d'authentifier la clé à télécommande, la communication nécessiterait plusieurs messages.

Un identifiant dans le porte-clé est lu sans fil, par exemple, via le standard RFID (DA_1). Une base de données avec les préférences enregistrées pour le conducteur (pour cet identifiant) est lue (DA_2) sur l'unité principale. Selon ces préférences, la position assise souhaitée est envoyée au contrôleur de siège (DA_3), où plusieurs servomoteurs sont déclenchés pour arranger le siège (DA_4). Le scénario est représenté à la figure A.8. Un diagramme de séquence est présenté á la page 45, figure 3.7.

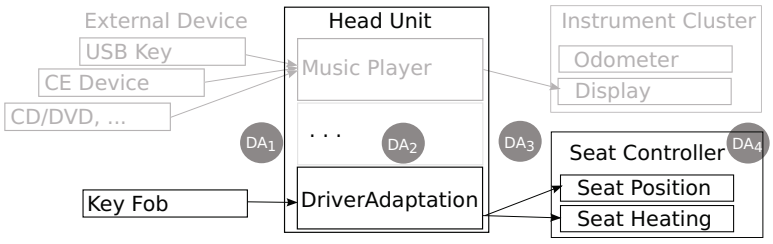


Figure A.8: Scenario III : Les sièges sont automatiquement ajustés en fonction de la clé utilisé pour déverrouiller/démarrer le véhicule.

A.4 Sécurité des Communications

En termes d'assurance vis-à-vis du fonctionnement des automobiles, la communication entre les différentes unités est primordiale. Aucun véhicule ne serait plus en mesure aujourd'hui de démarrer ou de rouler sans qu'au moins quelques composants électroniques de base (par exemple, le contrôle moteur et les systèmes de freinage) communiquent par les réseaux (des bus) internes.

Comme les systèmes du véhicule sont plus facilement accessibles, par exemple, en connectant les appareils grand public via USB ou Bluetooth, mais aussi avec la mise en place d'Ethernet dans les architectures actuelles, le réseau de bord devient une cible de choix pour les attaquants.

Pour de futures applications Car2X coopératives, qui sont en cours de normalisation en ce moment [Eur09b], la confiance dans les données reçues est très importante. Dans les approches actuelles, seule la liaison sans fil entre véhicules, par exemple i.e., Car2X avec la norme IEEE 802.11p, est définie comme sécurisée par des signatures cryptographiques. L'intérieur du réseau de bord, tout aussi important afin de garantir la sécurité de bout en bout, n'a pas été spécifié.

Dans ce chapitre, nous abordons cette nécessité de sécuriser la communication entre les différentes unités du réseau de bord. Nous décrivons i) un protocole de distribution de clé et ii) un protocole de transport sécurisé pour le réseau embarqué. Les protocoles de communication sont conçus en gardant à l'esprit les limitations dues aux contraintes des bus et aux performances de communication de l'architecture informatique.

Nous avons conçu les protocoles et les interfaces de ce protocole de façon à ce qu'ils s'intègrent d'une part avec les Hardware Security Modules (HSMs) conçu dans le projet EVITA, et d'autre part avec le cadre de la sécurité des logiciels comme un prototype. L'intégration des fonctionnalités de sécurité dans un cadre logiciel est importante comme l'architecture et la conception fonctionnelle du réseau d'un véhicule est en grande partie dirigée par les modèles. Dans une approche pilotée par les modèles comme c'est le cas dans AUTOSAR, le cadre assure que les fonctions du logiciel peuvent être déployées n'importe où dans le réseau.

Comme la communication est un élément central pour presque toutes les fonctions automobile qui s'étendent entre plusieurs ECUss, les techniques présentées dans ce chapitre s'appliquent à l'ensemble de nos trois scénarios.

A.4.1 Distribution de Clés dans le Système Embarqué

L'un des principaux éléments de communication intégré sécurisé est la *gestion des clés*. Il est essentiel d'utiliser des cés fraîches qui sont redistribuées périodiquement. Si seules des clés statiques devaient être utilisées, les attaques

contre ces clés (jamais modifiées) permettrait de définitivement casser la sécurité du système. Nous utilisons des clés statiques uniquement dans le but d'assurer le chiffrement des transports et l'authentification de clés, c'est à dire qu'elles ne sont jamais directement utilisé pour chiffrer ou authentifier des données en dehors des clés de session fraîchement générées et ne quitteront jamais le HSM en dehors du processus d'appariement initial qui se déroule à l'usine, dans un environnement de confiance.

La méthode présentée utilise la sécurité matérielle en vue d'améliorer les performances par l'utilisation de la cryptographie symétrique, tout en maintenant un contrôle d'accès sur la clé elle-même.

A.4.1.1 Utilisation asymétrique de clés symétriques

Nous utilisons des secrets partagés (les clés symétriques) en raison de contraintes de coût et de déploiement intégré automobile. Le déploiement de ces clés nécessite un dispositif ou module matériel de sécurité, le HSM, associé à chaque ECU. Le HSM met en œuvre des primitives cryptographiques accélérées et assure également le stockage sécurisé des clés, empêchant ainsi qu'une attaque logicielle réussie ne compromette notre infrastructure de clés. L'utilisation du HSM assure aussi la médiation de l'accès aux clés cryptographiques. Les clés possèdent des champs de données supplémentaires, notamment une date d'expiration, un code d'authentification et des paramètres d'usage (*use-flags*). Ces paramètres d'usage permettent de distinguer le HSM pour lequel la fonctionnalité d'une clé spécifique peut ou ne peut pas être utilisée. En règle générale, le HSM fera respecter certains usages sur les clés symétriques qu'il stocke, sur la base de ces paramètres : par exemple, une clé symétrique peut être étiquetée pour la signature (par exemple, la génération d'un MAC) pour un HSM alors que cette même clé réside dans plusieurs autres modules de sécurité matérielle, où elle est étiquetée seulement pour la vérification de la signature (en réalité l'authentification du message). Ce contrôle de l'utilisation permet d'assurer une utilisation asymétrique des clés, tandis que le calcul lui-même est efficace sur la base de la cryptographie symétrique.

Definition: La clé $k_{n,g}$ est la clé de session utilisée pour générer un Message Authentication Code (MAC) à ECU_n (l'entité émettrice) et la clé $k_{l,v}$ pour la vérification des données reçues (y compris la vérification du MAC) à ECU_l (l'entité de réception).

Deux autres indicateurs sont utilisés dans ce chapitre : les indicateurs d'$exportation$ et de $transport$. Le premier permet à une clé d'être exportée sous forme chiffrée et le second permet à une clé d'être utilisée pour chiffrer une autre clé pour l'exportation (cette autre clé doit être exportable). Une clé cryptographique ne peut en aucun cas quitter le HSM en clair. Pour les clés de session, la partie

vérification ($k_{x,v}$ for ECU_x) doit être marquée exportable pour la distribution des clés. En outre, une clé d'authentification distincte est utilisée pour valider une clé exportée et le transport chiffré.

Pour chaque ECU$_n$ qui établit un canal sécurisé avec d'autres ECUss, il y a une clé de chiffrement pour son transport k_n, t, et une clé d'authentification k_n, a, déployées sur son propre HSM et sur le HSM du gestionnaire centralisé des clés ou nœud maître (Key Master KM).

A.4.1.2 Distribution dynamique des clés

Partager globalement des clés compromettrait la sécurité globale du système et est évité dans notre solution. La répartition des clés entre les partenaires de communication se fait à travers une entité appelée KM. Chaque ECU e_i sur le réseau interne du véhicules partage deux clés secrètes avec le KM. La première clé $k_{i,a}$ est utilisé pour $authentifier$ l'ECU au regard des autres calculateurs, tandis que la seconde clé $k_{i,t}$ est utilisé pour le $chiffrement$ de transport des clés générées. Un nœud maître (KM) est une entité logique qui peut être présente sur un ou plusieurs calculateurs. Il est possible d'avoir plusieurs KMs (par exemple, un pour chaque domaine), de sorte que les clés ne sont partagés que par un nombre limité de fonctionnalités. Pour expliquer le protocole de distribution de clés, nous avons simplifié le déploiement avec seulement un gestionnaire centralisé ou nœud maître (KM pour Key Master) présent dans le réseau du véhicule (voir figure A.9.). Pour plusieurs KMs, une liaison supplémentaire entre les maîtres impliqués doit être établie avec le protocole de distribution de clé décrit dans la section 4.1.5. Dans ce cadre, un système distribué pour la politique gestion d'autorisation de connexions des différents domaines du véhicule est utilisé [SIR+10].

La figure A.9 représente l'architecture de gestion des clés. Les détails des phases de distribution sont exposées dans la figure A.10.

A.4.2 Communication Securisée sur le Bus CAN

Le bus de communication le plus couramment utilisé dans les véhicules d'aujourd'hui est le bus CAN qui est exploité à 500 kbit / s et dispose de 8 octets de données de charge utile (payload) par paquet. Du point de vue de la sécurité, une tranche de huit octets pour la charge utile de sécurité ne fournit pas une protection adéquate. D'un autre côté, des données plus grandes ou un champ dédié à la sécurité n'est pas possible, car le protocole ne peut pas être changé sans modification de tous les émetteurs-récepteurs CAN sur un bus. Il existe une approche rétro-compatible permettant d'incorporer une charge utile de sécurité en augmentant le taux d'échantillonnage, qui a été validée dans un FPGA mis

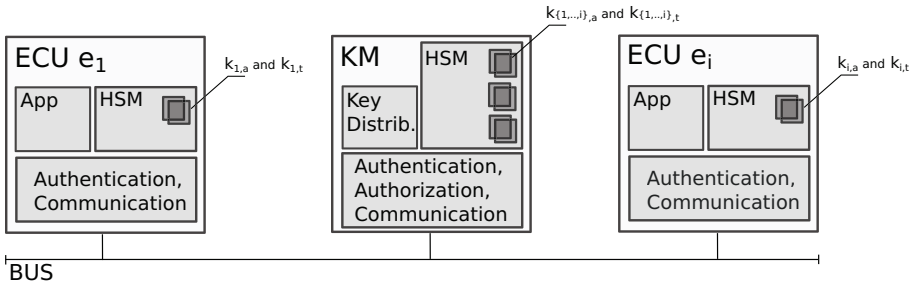


Figure A.9:
Un ECU agit en tant que nœud maître, c'est à dire gestionnaire de clés (KM pour *Key Master*). Les clés symétriques sont partagés avec tous les autres ECUs (représenté par e_1, e_i). Toutes les clés se trouvent dans le HSM correspondant (par exemple, utilisé pour l'authentification). Les politiques d'accès à l'ensemble du système sont évaluées par le nœud maître (Autorisation).

en œuvre par [HSV11]. Toutefois, il n'est pas possible de changer la conception de tous les émetteurs-récepteurs CAN dans un avenir proche, compte-tenu également du fait que des bus alternatifs existent déjà. Dans ce qui suit, nous montrons comment nous utilisons un protocole qui a été principalement utilisé pour le diagnostic du véhicule, le protocole de transport ISO-TP [ISO04], afin de transmettre de plus grandes charges utiles sur les bus CAN existants.

Nous générons des MACs afin d'authentifier les données transmises. Nous pouvons ajuster la longueur du code d'authentification afin de tenir compte de la latence et de la charge du bus. Les détails concernant ce protocole peuvent être trouvés à partir de la page 67.

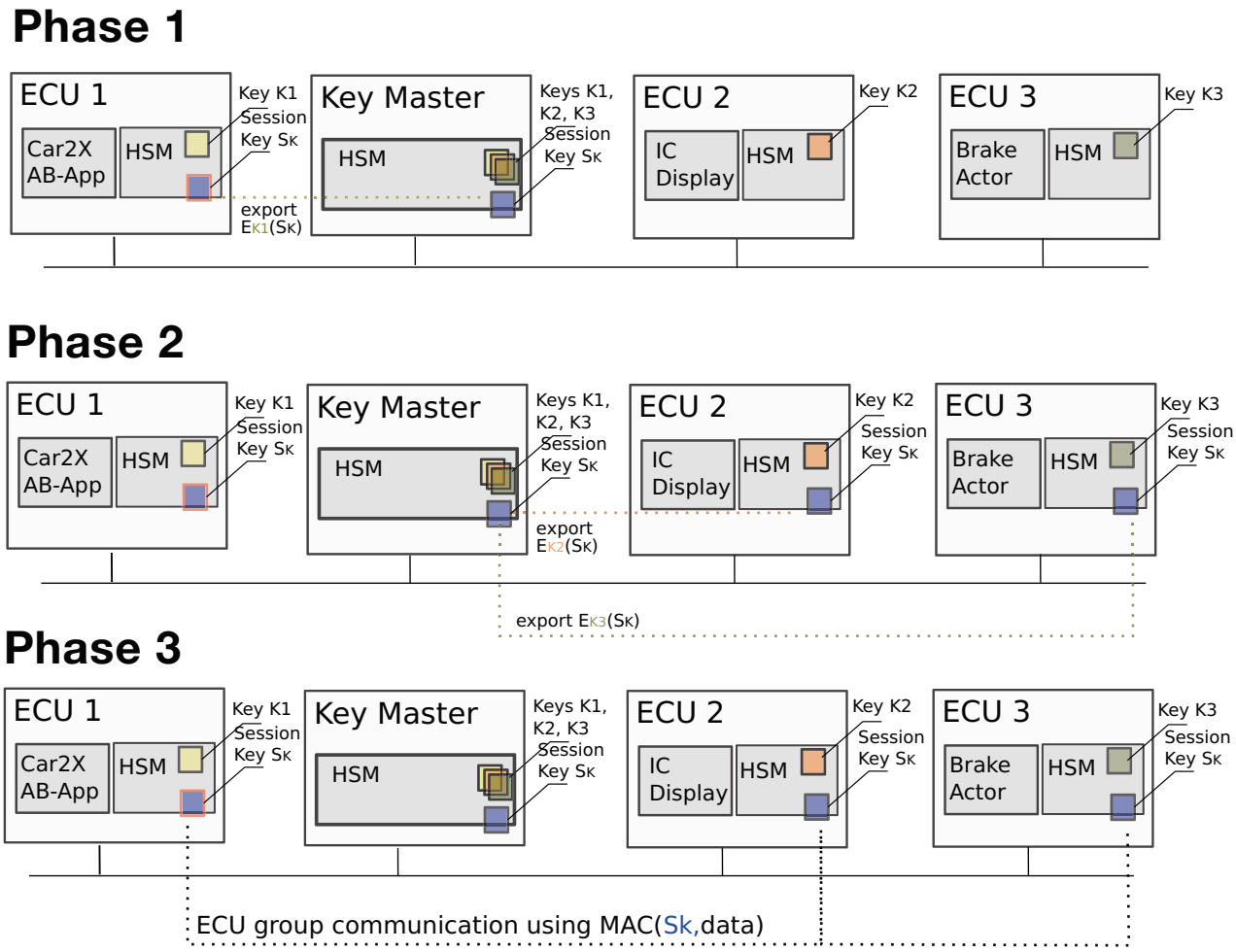


Figure A.10:
Les phases du mécanisme de distribution de clés : une clé de session ECU_1 (en bleu) est générée à l'initialisation (phase 1). Une clé de session $ECU_{\{2,3\}}$, permettant uniquement la vérification (sans marge rouge) est exportée vers le nœud maître, où elle est distribuée à tous les membres du groupe, (phase 2). Après la distribution réussie des clés (phase 3), les données envoyées à partir de ECU_1 peuvent être authentifiées par tous les membres du groupe. L'ECU décrit a été utilisé pour le véhicule de réception dans le cas d'utilisation de freinage actif (scénario I)

A.5 Sécurité Dynamique des Plate-formes

Dans ce chapitre, nous décrivons l'approche que nous avons suivi pour améliorer la sécurité de la plate-forme de calculateurs embarquées d'une automobile au-delà de la seule communication sécurisée. Elle diffère considérablement des approches statiques. Alors que le cas statique se suffit d'un processus d'amorçage (bootstrap) sécurisé et de signatures de logiciels, ces éléments ne peuvent garantir une résistance contre l'exploitation de vulnérabilités du logiciel s'exécutant sur un calculateur. Nous avons expliqué les techniques de ces blocs de construction dans le chapitre 2.

De nombreuses attaques réelles effectuées sur des véhicules du commerce ont été démontrés dans [CMK+11], [KCR+10], et [RMM+10]. La plupart ne se sont pas appuyées sur des failles des mesures de sécurité statiques mais sur l'exploitation de vulnérabilités du système d'exécution, par exemple, le débordement de variables dans un autoradio ou des failles de l'appairage Bluetooth et de la modulation radio CDMA. Ces travaux ont également montré la possibilité d'attaques contre des mesures de protection classiques comme le déchiffrement du code d'accès à deux octets pour la mise à jour de micrologiciels. Cette attaque a permis de reprogrammer un ECU avec un micrologiciel modifié. En résumé, ces exemples d'attaques suggèrent que des moyens de protection au-delà de la sécurité des communications et des approches purement statiques sont requis.

Nous avons employé une architecture générique pour réaliser un système embarqué de détection d'intrusion et qui est décrit dans la première section du chapitre (5.1) qui se trouve sur la page 77.

Une combinaison de m'ecanismes de suivi et de surveillance des flux de données sous-tend le concept de notre IDS. Ce mécanisme s'appuie sur le marquage des données au niveau local à l'intérieur d'un ECUs, mais aussi par la propagation de ce marquage entre plusieurs ECUss afin de contrôler les flux d'informations entre les applications locales et les applications distribuées. Ce mécanisme nécessite des ECUs avec une capacité de traitement assez importante. Nous proposons un deuxième mécanisme, simplifié pour des ECUss moins puissants, mais avec des risques de faux négatifs ou de faux positifs plus significatifs.

A.5.1 Protection de la Confidentialité et de la Sécurité par le Suivi de Flux d'Information

Dans le véhicule, les données sont échangées en permanence entre les différentes unités via les réseaux embarqués. Alors que la communication entre les capteurs, les contrôleurs et les actuateurs constitue plus ou moins un système fermé, l'intégration d'appareils grand public comme les téléphones mobiles et notamment les smartphones et la possibilité de télécharger de nouvelles applications crée un environnement plus dynamique. Ces nouvelles fonctionnalités exigent

un accès aux services des véhicules, qui ont été historiquement isolés de toute communication externe. En outre, les véhicules pourront bientôt recevoir des données provenant d'autres véhicules ou de l'infrastructure routière via la communication Car2X. Ces données peuvent être forgées de toute pièces à des fins malveillantes telles qu'exploiter les vulnérabilités des piles de protocoles du véhicule.

Nous tenons à souligner deux types d'attaques qui peuvent aller au-delà des capacités de mesures de sécurité statique : i) le détournement du flux de contrôle d'une requête protocolaire et ii) l'obtention et l'utilisation de données personnelles pouvant porter atteinte à la vie privée du conducteur.

Conditions préalables Deux conditions préalables doivent être remplies afin de s'appuyer sur ces données marquées distribuées : i) leur transmission doit être sécurisée et ii) la sécurité statique de la plate-forme doit être assurée pour tous les nœuds dans la séquence de traitement. Ces conditions sont remplies par l'utilisation des protocoles de communication sécurisés développés dans le chapitre A.4 et du Hardware Security Module d'EVITA décrit dans la section A.3.1, que nous utilisons comme des blocs de construction.

Pour être en mesure de combiner et de propager les informations de marquage sur place, un moteur de marquage dynamique de l'information est utilisé. Il propage des balises ou étiquettes de marquage dans les instructions en cours de traitement. Contrairement à une plate-forme d'exécution unique les nœuds intermédiaires de traitement doivent s'assurer de l'authenticité et de l'intégrité des étiquette de marquage transmises en propageant l'information aux calculateurs distants.

En plus des menaces sur la vie privée liées à la fuite d'informations, le contrôle de flux d'information qui est analysée au niveau d'instructions processeur uniques peut également servir à protéger l'exécution contre l'exploitation de vulnérabilités telles que les attaques par débordement de tampons (buffers) et du tas (heap). Dans de tels cas, le marquage de contrôle des flux de données est vérifiée avant un appel (par exemple, les appels *return* ou *free*), ce qui permet de repérer l'attaque.

A.5.1.1 Suivi dynamique de flux d'information

Les programmes et applications véhiculaires sont organisés autour de la communication. Ils sont construits en petits blocs fonctionnels suivant un modèle de composants qui définit le raccordement des entrées et sorties des modules logiciels de base ou composite. Ces modules logiciels sont finalement placés sur des calculateurs différents, générant ainsi des modes de communication spécifiques. Des intergiciels tels qu'AUTOSAR pour l'automobile sont généralement utilisés pour déployer de telles architectures.

Dans un tel système distribué, qui associe logiciels et matériels, il est important de suivre la circulation des données non seulement à l'intérieur, mais aussi entre les calculateurs. Par exemple, une attaque réussie par le biais du système de surveillance de la pression des pneus comme indiqué dans [RMM+10], dans lequel les capteurs transmettent leurs données sans authentification par un réseau sans fil, il est évident que des données apparemment inoffensives peuvent compromettre des parties du véhicule, comme par exemple l'affichage de l'indicateur de vitesse. Des données comme la valeur d'un capteur de pression des pneus sont également couramment diffusées sur le réseau de bord du véhicule et distribuées à des calculateurs spécifiques. Ces données peuvent être utilisées par des fonctions de contrôles de plausibilité dans le module de contrôle électronique de la stabilité du véhicule ou pour déclencher la fonctionnalité "limp home" (dans le cas d'un pneu à plat) dans d'autres unités. Si les données d'origine ont été construites de manière malveillante, un calculateur utilisant ces données pourraient faire l'objet d'un exploit par lequel un attaquant peut prendre le contrôle de son exécution en détournant le flot de contrôle normal de l'unité centrale. Un débordement possible dans un dispositif comme le contrôle électronique de stabilité peut se traduire par une défaillance du système de freinage, car ce calculateur contrôle directement et individuellement les actionneurs de frein de chaque roue.

Une architecture multi-couches L'approche multi-couches de notre architecture est représentée sur la figure A.11 : Notre approche s'appuie sur la surveillance de la circulation de l'information provenant du réseau et des processus internes. Celle-ci est introduite au niveau de l'intergiciel qui gère les échanges d'information. Les données entrantes sont étiquetées par un marquage que nous introduisons. Nous vérifions la présence et l'absence d'étiquettes pour les données sortantes en fonction d'un ensemble de règles définissant la politique de sécurité. Une partie importante de ce système est le moteur de marquage basé sur l'instrumentation binaire : toutes les applications et les bibliothèques sont instrumentées à l'exécution. Le marquage se propage à l'exécution parmi les registres et la mémoire de l'ECU. Comme les différents types de sources de données en entrée doivent être surveillées, nous utilisons des marques distinctes, une approche souvent appelée " marquage' de couleur" [EKS+10].

A.5.1.2 Instrumentation binaire pour le suivi des balises de marquage

Afin d'introduire des marqueurs dans les applications en cours d'exécution, nous utilisons une technique appelée "instrumentation binaire". Cette technique injecte du code personnalisé pour les binaires au moment de l'exécution, c'est à dire que l'on peut instrumenter les instructions machine et les appels système afin de suivre le flux de données entre les registres ou les zones de la mémoire, ainsi que prendre des précautions si les données sont utilisées d'une manière

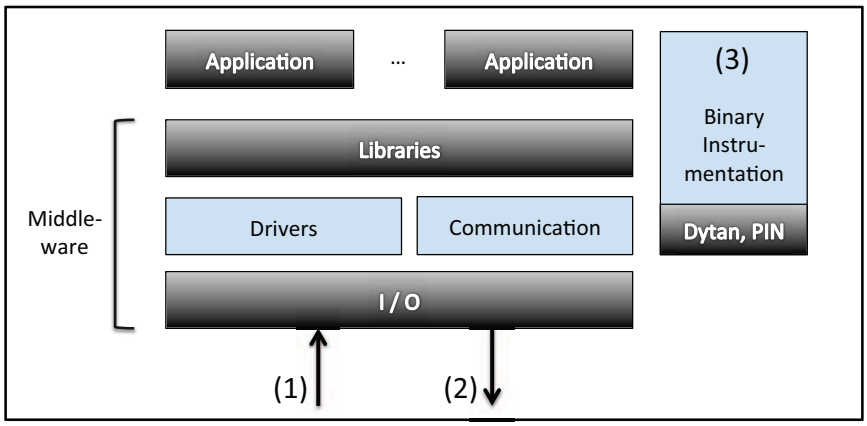


Figure A.11: L'architecture multi-couche. Les applications utilisent les bibliothèques basées sur des pilotes et fonctions de communication de la plate-forme pour échanger des données. Lorsqu'elles arrivent dans la couche du pilote et de la bibliothèque de communication, les $données\ qui\ sont\ lues$ (1) sont étiquetées avec un marquage approprié, par exemple $sensor_n$. L'étiquette de marquage est maintenue dans une mémoire fantôme (shadow) et propagé tout au long de l'exécution. Dytan instrumente les binaires applicatifs et les bibliothèques à l'exécution, et propage le marquage dans la mémoire (3). Lorsque $les\ données\ sont\ écrites$ (2), les règles de la politique de sécurité sont appliquées, c'est à dire que l'on analyse si les données ne contiennent que des informations provenant de sources autorisées, à savoir que les étiquettes de marquage correspondent à celles attendues dans les règles.

douteuse. L'un des principaux avantages de l'instrumentation binaire est que le code source des programmes analysés n'a pas besoin d'être disponible. Cela permet de s'assurer que des programmes d'origine inconnue respectent des règles établies. Nous utilisons l'outil Pin d'Intel [LCM+05], combiné à un moteur de marquage. Pin est un outil générique pour l'instrumentation binaire dynamique et fournit la base pour l'analyse dynamique de marquage. Pin est disponible pour les architectures x86 (Windows, Linux, Mac) et aussi pour ARM. Nous avons mené des expériences sur IA32 Linux.

Il existe un certain nombre d'implantations de prototypes pour le suivi dynamique de flux de données basées sur Pin [CLO07, KPJK12, ZJSK11]. Nous avons analysé la faisabilité d'appliquer ces implantations prototypes dans notre cadre de sécurité. Nous décrivons l'implantation réalisée pour tracer des données lors de l'exécution sur la plate-forme et pour propager les marqueurs avec la communication entre les calculateurs embarqués dans la voiture aux chapitres 3 et 4.

A.6 Conclusion et Perspectives

Nous affirmons que nos approches augmentent le niveau de sécurité des réseaux véhiculaires embarqués en appliquant la cryptographie pour la communication, l'utilisation de techniques de marquage pour le réseau du véhicule, ainsi que l'application stricte de règles dans le cadre du système de détection d'intrusion. Notre prototype de démonstration et nos implantation sont basées sur un cadre de sécurité automobile réel. Nous avons simulé et testé la faisabilité de ces approches sur des réseaux embarqués actuels. Même sur des réseaux restreints, tels que le bus CAN, notre approche de communication sécurisée est viable et semble avoir été prise en compte par le développement industriel [KB11]. Les protocoles de distribution de clés et de communication sécurisée ont été mis en œuvre avec succès pour des applications Car2X à l'intérieur de deux véhicules BMW au Forum Car2Car de Novembre 2011 à Erlensee, en Allemagne.

Les contributions qui ont été présentés dans cette thèse peuvent être mieux illustrées par la comparaison suivante. Une publication fréquemment cité sur la sécurité embarqué, "*sécurité dans les systèmes embarqués : les défis de conception*" [RRKH04], nomme six exigences en matière de sécurité pour les systèmes embarqués. Sur la base de ces exigences génériques, nous comparons l'architecture des véhicules d'aujourd'hui, avec les améliorations suggérées par les contributions de cette thèse. La comparaison détaillé se trouve au Chapitre 7, à partir de la page 111.

Ces exigences sont :

- L'identification/authentification, l'accès au réseau, et la sécurité de la communication,
- Le stockage, la sécurité des contenus et la disponibilité des systèmes
- Exigences supplémentaires : La protection de la vie privée et la détection et limitation des attaques

Discussion sur la faisabilité de l'approche proposée Nos solutions pour sécuriser l'architecture à bord de véhicules ont été mises en œuvre et évaluées sur un prototype de démonstration. Un déploiement en production a cependant des exigences beaucoup plus strictes. Le coût est un facteur important dans le monde de l'automobile : si les composants du véhicule coûtent $50 de plus (imaginez un HSMs dans plusieurs ECUss), et que le véhicule est vendu à dix millions d'exemplaires, le manque à gagner se monte déjà à un demi-milliard de dollars. Les clients ne sont également pas prêts à payer pour des fonctionnalités de sécurité, car ils s'attendent à ce que le produit soit déjà sécurisé tel quel.

Si le coût est une préoccupation, l'intégration dans les modèles de communication et la puissance limitée de traitement des composants actuels en sont

d'autres aussi importantes. Nous avons montré que la surcharge de communication pour les protocoles de communication sécurisés est limitée. Néanmoins, une latence supplémentaire est introduite.

L'intégration des protocoles cryptographiques avec les systèmes d'exploitation automobile n'est pas un facteur anodin non plus, car de nombreux systèmes différents existent, comme par exemple OSEK, QNX, VxWorks, ou même Windows Embedded Automotive. Les composants de faible coût tels que les capteurs n'ont souvent pas à proprement parler un véritable système d'exploitation et utilisent un modèle de communication statique combinée avec une machine d'état. L'approche d'AUTOSAR avec un intergiciel standard répond à ce problème. Cependant, un déploiement à grande échelle et à faible coût des composants est encore une perspective lointaine.

Le suivi des flux d'information en utilisant une instrumentation du code compilé a le grand avantage de pouvoir être appliqué sur des composants logiciels au code source duquel on n'a pas accès. Cette approche a cependant une grande influence sur la performance de l'exécution des programmes. Naturellement, la compilation source-à-source pour le suivi des informations, comme l'illustrent les systèmes GIFT [LC06b] ou DIVA [XBS06], fournit une bien meilleure performance (entre 5% et 50% de surcoût), mais elle repose sur la disponibilité du code source et sa recompilation. Ce n'est pas le cas pour le développement de composants logiciels par des tiers dans des environnements d'exécution ouverts ou dans le cas des magasins d'applications en ligne. Une meilleure performance pourrait être atteinte par l'utilisation d'un matériel dédié au suivi de marquages de sécurité, ou par une recompilation du code intermédiaire, comme pour l'environnement d'exécution Java ou un environnement de virtualisation.

Même si des mécanismes logiciels et matériels permettent d'accroître la sécurité du réseau embarqué, toute attaque réussie sur l'unité Car2X, qui évalue l'information interne et déclenche des alertes externes, rend vulnérable et compromet potentiellement l'ensemble du système de communication inter-véhiculaire. Il s'agit d'un problème général, aucune sécurité de bout-à-bout – à partir du capteur d'une voiture à l'actionneur d'une autre voiture – n'étant réalisable sans l'utilisation de la coûteuse cryptographie asymétrique et de grands efforts d'harmonisation entre les fabricants. Les différentes implémentations de mécanismes de sécurité à l'étude peuvent aussi constituer un obstacle supplémentaires pour un déploiement mondial : par exemple, les spécifications américaines, européenes et asiatiques ne sont pas interopérables et entraînent par ailleurs des restrictions à l'exportation du fait de l'utilisation de cryptographie.

A.6.1 Perspectives en recherche et développements futurs

Nous avons vu que le domaine du véhicule automobile n'a pas pris en compte la nécessaire mise en oeuvre de mesures de sécurité pour l'ensemble de l'architec-

ture embarquée. Certaines mesures de sécurité ont été appliquées uniquement à des interfaces exposées, comme par exemple le signal radio d'un porte-clés pour déverrouiller le véhicule qui commencent à intégrer des algorithmes de défi/réponse à distance bornée. Bien qu'une protection complète de toutes les unités embarquées et de la communication n'est pas réaliste pour les années à venir, la sécurité constitue indubitablement un sujet de recherche actif.

Sécurité de la communication embarquée Actuellement, même des fonctionnalités essentielles d'une automobile ne sont pas correctement sécurisées. Des outils pour le vol de voiture comme [Jac12] sont en vente libre. Ils se connectent à des câbles du réseau interne (qui peuvent être atteint depuis l'extérieur du véhicule, par exemple à travers l'espace entre la portière et la carrosserie), désactivent l'alarme du véhicule, et ouvrent les fenêtres. Cela rend aujourd'hui flagrant le manque de sécurité dans les communications dans le véhicule.

La version v4.0 de l'API d'AUTOSAR introduit des services cryptographiques dans sa spécification [BFWS10], ce qui facilite l'utilisation de mécanismes cryptographique dans le logiciel. Mais cette spécification ne définit aucun canal de communication sécurisé entre différentes unités fonctionnant sous AUTOSAR. Nous avons été agréablement surpris de constater qu'un protocole inspiré du protocole de transport sécurisé pour CAN que nous avons proposé était sur le point d'être mis en œuvre pour un produit industriel sous AUTOSAR [KB11].

L'évolution des réseaux embarqués vers Ethernet et IP est déjà en cours en ce qui concerne certaines parties de l'architecture embarquée [Bru10, Jon10]. L'utilisation d'Ethernet pour l'ensemble du réseau de bord est actuellement à l'étude [GHM+10]. Avec une bande passante et une taille de paquet supérieures, l'utilisation du protocole IP offre beaucoup des possibilités pour la sécurité. Tout en étant plus coûteux, il permet d'établir et d'appliquer des solutions de sécurité éprouvées dans les réseaux informatiques, comme le protocole IPsec.

Détection et réaction aux intrusions La détection des nœuds qui se conduisent anormalement et des paquets malformés est plus facile que dans les environnements réseaux normaux, parce qu'un véhicule est un système plutôt statique avec des modèles de trafic qui ne changent pas de manière significative au cours de l'exécution, par rapport aux réseaux d'ordinateurs. La détection d'intrusion a été étudiée pour le réseau embarqué d'une automobile dans [LNJ08, MG09, MA11], et constitue une partie de cette thèse et du projet EVITA [WWZ+10]. Les techniques d'apprentissage sans surveillance qui sont devenues populaires pour les systèmes IDS pourront probablement être utilisées dans le futur. La réponse à l'intrusion constitue un aspect du problème qui n'a pas été évalué de manière suffisante dans le cadre automobile : quel est le comportement adéquat en réponse à la été détection d'une menace avec un degré donné de confiance ? Est-ce

que certains composants du véhicule peuvent êtres éteints ou désactivés ? Ne risque-t-on pas de créer de nouvelles possibilités d'attaque par déni de service ? La notification d'une intrusion par un avertissement *"vous avez été piraté"* est-elle acceptable ou risque-t-elle de se révéler gênante voire dangereuse pour le conducteur ? Une publication de 2008 [HKD08a] fournit une classification des réponses possibles à des notifications d'alertes haptiques. Alors que la multimodalité est utilisée pour de nombreuses fonctions du véhicule, nous sommes sceptiques concernant son utilisation dans le cadre d'une attaque informatique, qui consisterait par exemple à ''influencer la force sur la pédale de frein ou le volant", comme la publication le décrit. La question de savoir comment réagir intelligemment à une détection d'intrusion est, à nos yeux, encore un problème non résolu et difficile — surtout lorsque seuls des informaticiens se penchent sur le problème. La conception d'interfaces homme-machine HMI et sur l'ergonomie doivent être considérés dans la solution de ce problème.

Suivi dynamique de flux d'information La surveillance de l'usage des données dépasse le simple contrôle d'accès. Si le suivi dynamique de flux d'information (ou DIFT) a été utilisé depuis un certain temps dans l'analyse de logiciels malveillants, son utilisation pendant l'exécution d'un logiciel est assez nouvelle, et il n'a que récemment été appliqué aux systèmes embarqués et distribués [EGC+10, ZPK11]. Cette technique est particulièrement intéressante pour le monde de l'automobile, où le logiciel embarqué est en train d'évoluer d'un développement en interne vers la mise en place de "boutiques d'applications en ligne" pour le domaine de divertissement — d'une manière équivalente à ce qui s'est déroulé pour les smartphones ces dernières années. L'ouverture aux développeurs de l'AppStore R-Link de Renault est prévue pour Novembre 2012 [Lut11]. Même si un processus d'examen rigoureux, comme celui imposé par Apple (voir section 2.3.2), offre un certain niveau de sécurité, il ne peut jamais fournir une assurance complète. Des applications malveillantes peuvent se glisser au travers des mailles du filet [Sli12], et des défauts dans les applications logicielles peuvent entraîner des vulnérabilités exploitables à l'exécution, comme indiqué dans la section 2.3.3.

Une approche DIFT similaire à la nôtre peut être appliqué à des applications inconnues, et ainsi prévenir les atteintes à la vie privée du conducteur comme [Sli12]. Le principal problème à résoudre est la performance, car l'instrumentation binaire implique un surcoût important à l'exécution, même lorsque des moteurs de marquage à faible empreinte sont utilisés. Une solution à ce problème pourrait être une approche combinée, dans laquelle les fonctions de confiance du code soientt validées et exemptées de ce suivi d'exécution de fine granularité. Nous travaillons actuellement sur une analyse de la faisabilité d'une approche qui combine un suivi basé purement sur l'utilisation d'un intergiciel (qui imlique peu ou pas de surcoût) avec une instrumentation binaire pour l'environnement du reste du véhicule.

À long terme, les logiciels malveillants peuvent s'adapter à ces techniques. Les systèmes de détection d'intrusion peuvent toujours être contourné par divers moyens. Par exemple, nous voyons les virus et les vers informatiques d'aujourd'hui détecter s'ils sont exécutés dans un environnement virtuel. Cela est également vrai pour les analyses dynamiques d'exécution. Egele et al. [ESKK08] mettent l'accent sur ce point dans leur enquête sur les techniques d'analyse dynamique de logiciels malveillants : *"Si les techniques de suivi de flux d'information deviennent prédominantes dans les outils d'analyse, on peut s'attendre à ce que les auteurs de logiciels malveillants trouvent des mécanismes qui essayent de contourner ces systèmes"*.

Sécurité matérielle L'usage d'un module matériel de sécurité (ou HSM) est l'un des principaux éléments techniques du projet EVITA. Ce module a non seulement été utilisé pour sécuriser les communications à bord, mais il a également été intégré à une unité de communication C2X simTD, dans laquelle il a assuré l'accélération matérielle des opérations cryptographiques à base d'algorithmes ECC utilisées pour protéger les communications intervéhiculaires [2]. Une suite au prototype FPGA est en cours d'élaboration par le projet européen PRESERVE [PRE14] qui devrait déboucher sur un composant ASIC sur la base des résultats d'EVITA. Un prototype est prévu pour être utilisé dans la phase finale du FOT français SCORE@F en 2013.

Le module matériel de sécurité SHE propose une approche similaire à celle du HSM d'EVITA. Une comparaison est donnée dans [WG11]. Ce module fournit également un amorçage (boot) sécurisé et l'accélération matérielle des algorithmes de chiffrement symétriques. Son cahier des charges a été convenu entre un certain nombre de constructeurs et équipementiers automobiles, le consortium HIS. Infineon et Fujitsu, qui faisaient partie du partenariat d'EVITA, ont annoncé qu'ils comptaient commencer la production de puces ASIC HSM pour l'automobile d'ici à 2014 [SS11]. Infineon prévoit également une intégration de ces modules avec leur ligne TriCore de microcontrôleurs automobiles comme le TC1797 utilisé dans EVITA.

Communication Car2X Pour la plupart des gens, "les voitures qui parlent" et se conduisent de manière autonome sont encore une vision extrêmement futuriste. Bien que la communication Car2X va se développer dans les années qui viennent, il existe déjà des normes préliminaires de l'IEEE et l'ETSI. Des tests d'interopérabilité et de conformité ont même déjà été définis et menés sur des prototypes en vertu d'un mandat officiel de la Communauté Européenne [Eur09b].

[2] En fait, les certificats RSA simTD n'ont pas été utilisés, mais des certificats de pseudonyme ECC ont été utilisés pour signer les données qui ont été échangés entre les deux véhicules de démonstration.

Nous sommes certains que la communication Car2X va faire partie du paysage des véhicules automobiles de demain. La sécurité des communications sera assurée par des signatures cryptographiques. Un module de sécurité matériel de bord sera selon toute vraisemblance requis par la norme pour assurer la sécurité, par exemple, des clés et du stockage des certificats ainsi que pour des raisons de performances, car les solutions purement logicielles ne peuvent pas soutenir la cadence des vérifications prévues. La prise de conscience croissante des attaques sur le réseau embarqué [CMK+11, KCR+10] est un indicateur fort que la sécurité jouera un rôle important dans la conception des futures interfaces inter-véhiculaires ainsi que dans les réseaux embarqués automobiles.

Appendix B

Additional Measurements and Implementation Details

B.1 HSM Performance Measurements

B.1.1 Performance Figures

A detailed performance analysis of the EVITA HSM that was used in the scope of this thesis has been published in [WG11]. The ECC signature generation and verification averages around 500 per second. This is important for the verification of Car2X Communication, especially in congested environments and at intersections, where a large amount of vehicles broadcast CAM frames at a high rate (up to 10 Hz per vehicle). This performance is adequate even for these dense scenarios, as predicted in [LLZ+08] and [G11].

B.1.2 Overhead of Prototype Implementation

The HSM prototype was implemented on a Virtex-5 FPGA board. The on-board PowerPC CPU was used to install an embedded Linux, which contains the firmware. This firmware uses standard TCP sockets on a dedicated ethernet interface to retrieve ASN.1 encapsulated HSM requests. While this allows agile development, the performance of this prototype was influenced by network and marshalling operations, i.e., the encapsulation of requests to a large extent.

To understand how the real processing time is distributed between network, ASN.1 coding and the actual HSM operations, we have instrumented the HSM firmware's source code with time measurement functionality. In Figure B.2 we show how the processing time is broken up between these tasks. Not surprisingly, for commands which involve I/O operations at the firmware level (e.g., import/export), the overhead is marginal (under 10%), while it is close to 80%

for key generation and cryptographic operations. This is the result of the slow flash memory that was used for storing data.

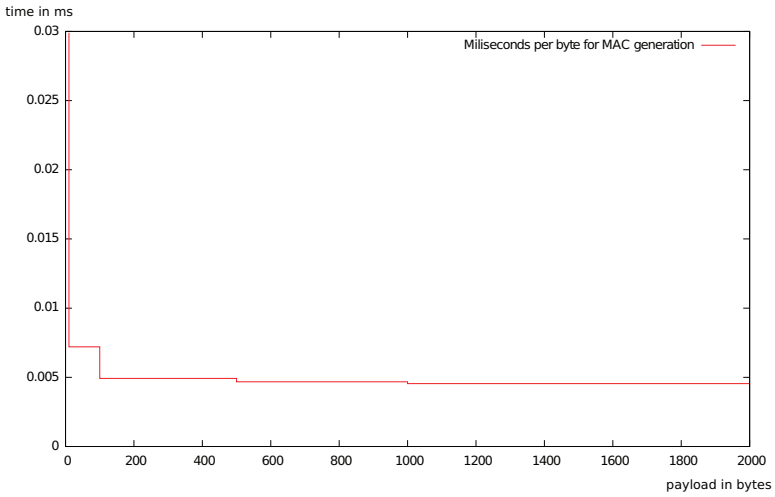


Figure B.1: Performance per-byte for generating a standard AES-128 based CMAC in the HSM. The time is given in [ms]. The input is padded to 32 bytes and varied between 10 and 2000 bytes.

The overhead of generating a MAC sinks with the amount of data that is hashed. In particular, all payload is padded to 32 bytes, resulting in poor per-byte performance for smaller payloads. The per-byte performance is shown in Figure B.1.

Out of all HSM operations, the import and export operations are most time intensive, due to I/O. Both import and export operation take around 15ms. This value can very likely be improved for key distribution, e.g., by holding temporary keys in volatile memory and never saving them to NVRAM.

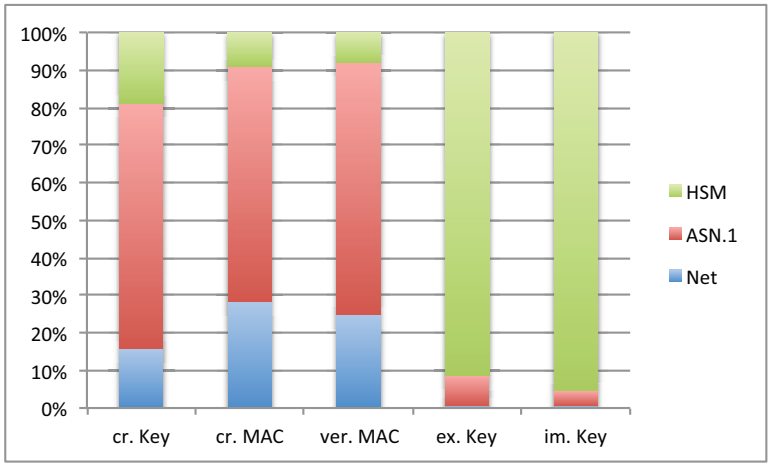


Figure B.2: Breakdown of the prototype processing time into network, ASN.1, and actual cryptographic HSM operations. The large difference is due to necessary I/O operations for import/export commands.

HSM function	net t	asn1 t	hsm	t/key	t/key (hsm)
create_random_key	243	1005	289	0.7685	0.14
export_key	256	2729	30879	16.932	15.44
import_key	255	1265	31297	16.409	15.65
create_MAC	1801	4971	597	3.68	0.30
verify_MAC	895	2404	291	0.22	0.018

Table B.1: Absolute and relative times of processing at the HSM prototype. The times are given in milliseconds. For the absolute time, a sample series of 2000 keys (or bytes for MAC) was used.

HSM function	net (%)	asn1 (%)	hsm (%)
create_random_key	15.84	65.42	18.81
export_key	0.76	8.06	91.19
import_key	0.78	3.86	95.37
create_MAC	24.45	67.46	8.1
verify_MAC	24.94	66.97	8.12

Table B.2: Relative percentages of individual HSM operations broken up into network, ASN.1, marshalling and actual processing time (cf. Fig. B.2).

B.2 Key Distribution Implementation

In the following, we give code examples or key distribution and secure communication implementations within the framework. We start with application code and present a simple example for sending and receiving data. We then show code excerpts from the client side and server side framework.

B.2.1 Application Code

The following listing shows an example for a client application that uses the framework. The actual communication is performed using the function `EMVY_secure_communication`. The program code used at the sender side is shown in listing B.1 and for the receiver side in B.2. If no keys had been exchanged before, the key distribution is triggered via the EMVY master before the payload is delivered.

Listing B.1: Application Code at Client Side (Sender).

```
2  static char GROUP_NAME[] = "group://group1";
3  static char TEST_PAYLOAD[] = "Das␣Pferd␣frisst␣keinen␣Gurkensalat.";

5  /* Init security property set */
6  memset(&security_property_set, 0, sizeof(SecurityPropertySetStruct));
7  security_property_set.authentic = TRUE;
8  security_property_set.confidential = TRUE;
9  security_property_set.fresh = FALSE;
10 security_property_set.integer = FALSE;

12 /* Init group entity */
13 memset(&group, 0, sizeof(Entity));
14 OCTET_STRING_fromString(&group.identifier, GROUP_NAME);
15 OCTET_STRING_fromString(&group.description, "no␣description");

17 /* Init payload */
18 memset(&payload, 0, sizeof(CommunicationPayload));
19 OCTET_STRING_fromBuf(&payload.data, TEST_PAYLOAD, sizeof(TEST_PAYLOAD)
       );


22 rc = EMVY_secure_communication(&group, &security_property_set, &
       payload);

24 if (rc != EvitaReturnCode_evitaReturnOk) {
25   LOG_WARN("EMVY_secure_communication␣failed␣with␣error␣code␣%d", rc);
26 }
```

Listing B.2: Application Code at Client Side (Receiver).

```
2 int process_group1_messages(OCTET_STRING_t * buf)
3 {
4   printf("Received␣message:␣%s\n", buf->buf,);
5 }

7 int main(int argc, char ** argv)
8 {
9   add_processing_function("group://group1", process_group1_messages);
10 }
```

B.2.2 Client Framework Code

The program code itself is rather lengthy as it involves retrieving configuration settings and initializing all possible key flags. Thus we show the programming interface by giving function signatures for secure communication and the key exchange in the following.

The communication itself is wrapped by the functions `EmvyKmmCom_down` and `EmvyKmmCom_up` that are called from within the framework. These wrapper functions use the previously exchanged session key to enhance the application payload with a security payload. This additional payload is added at the sender, and evaluated accordingly at the receiver side. The function `exchange_key_with_group` is called to trigger key distribution for a certain group. The key handle of the resulting session key is retained by the framework for future use by the wrapper functions.

If a node receives a session key through its Client-RPC interface, the request is first processed by the function `process_incoming_push_key_kmm_request` and then by `EmvyKmm_receiveGroupKey`.

For unicast communication, a response can be received through the payload buffer.

Listing B.3: Framework Program Code (Application/Client Side).

```
2 extern int EmvyKmmCom_down(
3        Entity* groupEntity,
4        const size_t dinSize,
5        const unsigned char* din,
6        size_t *doutSize,
7        unsigned char* dout );

9 extern int EmvyKmmCom_up(
10       const size_t dinSize,
11       const unsigned char* din,
12       Entity* groupEntity,
13       size_t* doutSize,
14       unsigned char dout[] );

16 enum EvitaReturnCode  exchange_key_with_group(
17       char * group_identifier,
```

```
18        SecurityPropertySetStruct * security_property_set);

20 static EvitaReturnCode process_incoming_push_key_kmm_request(
21        PushKeyKMM * pushKeyCmd);

23 extern EvitaReturnCode EmvyKmm_receiveGroupKey(
24         const char* groupIdentifier,
25         const size_t encryptedKeySize,
26         const unsigned char* encryptedKey,
27         const size_t authenticationTagSize,
28         const unsigned char* authenticationTag );
```

B.2.3 Server Framework Code

The following listing shows a shortened extract of the code on the server side. The server side is implemented in C++. The code excerpt has been rendered more readable by omitting some details. We want to show the distribution as the core function. The received request is first authorized, then group members are retrieved, the session key payload is authenticated and imported, and finally distributed to all group members in different threads.

Listing B.4: Framework Program Code (Server Side).

```
1 ReturnCode KMM::KMM_start_groupcom(EmvyServer * server, const Entity&
      endpoint,
2         UINT8 session_nonce,
3         const EMVY::KeyStruct & session_key,
4         const EMVY::Group & group) {

6   ...
7   ret = PDM::EMVY_request_authorization (endpoint, group, *operation,
        NULL);
8   if(ret->getCode() == 0)  {
9      authorized = true;
10     group_member_list = PDM::EMVY_retrieve_group_member(endpoint,
           group);
11  }
12  else {
13  LOG_WARN("PDM: Operation Denied. PDM did not authorize communication
        ") ;
14  SWD::SWD_log_event(new SecurityEventNotice(
15    "PDM did not authorize communication (in KMM_start_groupcom)",
16    this->ownEntity, new Timestamp(),
17    SecurityEventTypes_KMM_Auth_Fail));
18   }

20   ...
21   ret = key_import( transport_key_handle,
22             ... ,
23             (EXT_KEY_t *)session_key->getData(),
24             ... );


27   for (Entity::ConstIterator itr=group->getFirstGroupMember(); itr!=
         group->getLastGroupMember(); itr++)
28   {
29      evita::util::Thread * thread = new DistributorThread(
```

```
30    *itr,
31    group->getIdentifier().c_str(),
32    session_key_handle,
33    server);
34       thread->start();
35       threads.push_back(thread);
36    }
```

B.2.4 Detailed MSC for Key Distribution

In the following figure, you can see a detailed sequence chart for the key distribution protocol from Section 4.1. This chart was used as detailed specification in [SIR+10]. The abbreviations relate to implementation specifics: PSK is an ECU-specific Pre Shared Key for authentication and transport encryption, EAM is the Entity Authentication Module, and PDM refers to the Policy Decision Module of the framework. The `OpenChannel_REQ` message, is sent to the key master through its Server-RPC interface. The session key (SeSK) is then re-encrypted for all ECUs of the group and pushed to them via the Client-RPC interface `OpenChannel_CMD`.

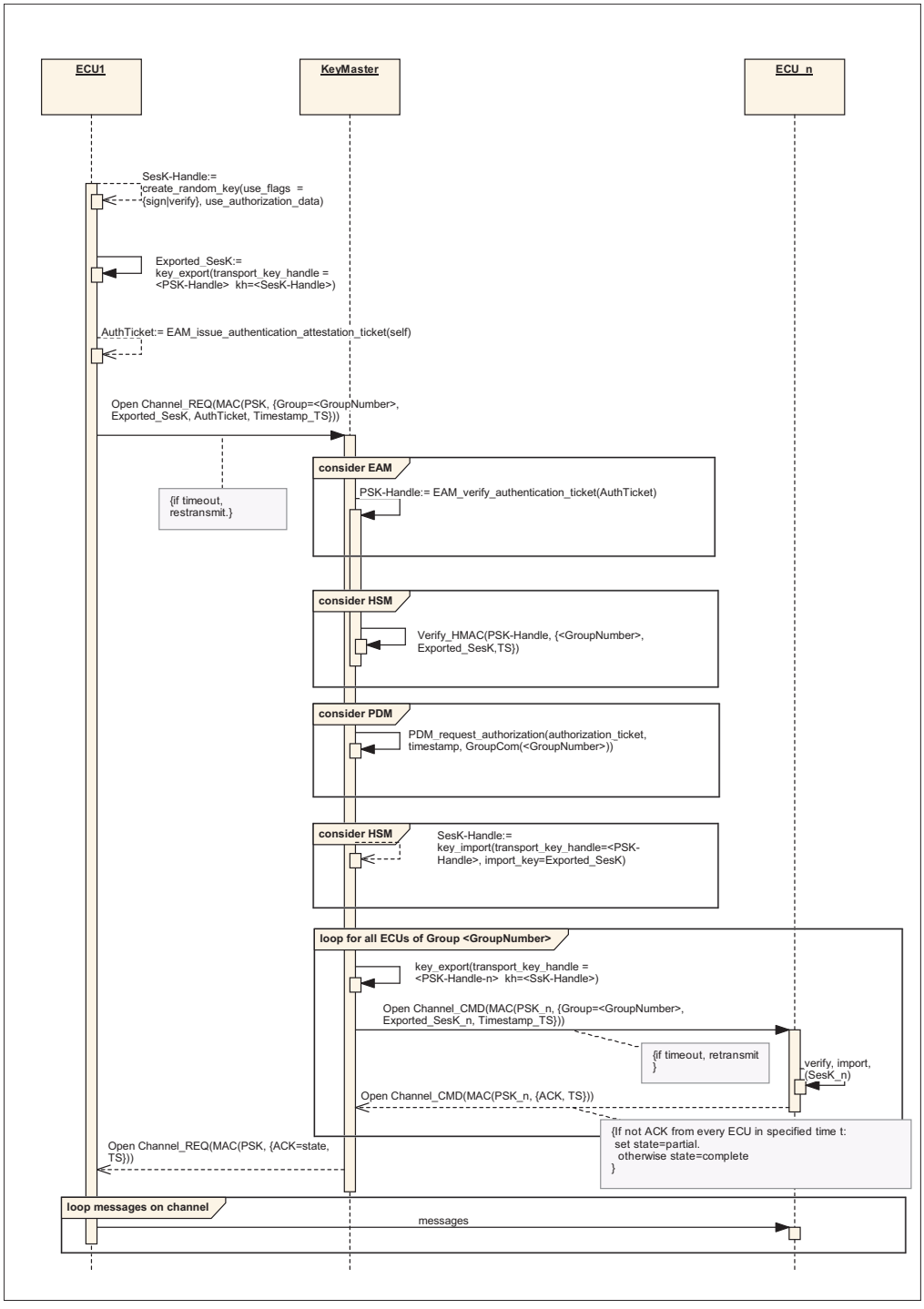


Figure B.3: Detailed Group Communication Establishment: A key pair is generated at the initiator, ECU_1. The sign-key can not be exported and stays inside the HSM. The verify-key is exported and transmitted via the KM to all other ECUs of the group.

B.3 Secure Storage

Certain data needs to be stored securely on a platform. In addition to access control, that is, *who* may access a certain resource, encryption can help to ensure the confidentiality of data. Encryption also holds if more sophisticated techniques to access protected data, such as reverse engineering contents of memory, are applied. Encryption can be used to protect sensitive content, such as system configurations, navigation data, or user content in order to provide secure storage.

In this Section, we show how a simple secondary encryption protocol can be employed together with the HSM and its platform register, in order to provide *Secure Storage*. In contrast to the specification, the implementation of Secure Storage has been integrated into the middleware framework by a partner of the EVITA project, and was thus not part of the thesis work.

B.3.1 Key Access Control

Depending on the kind of storage (user data, application data or system data), different measures are taken. Especially the cryptographic keys used for encryption need to be protected. The ECU Configuration Register (ECR) is a special register, which functions similar to a TPM's trusted platform registers: At certain points in the boot phase and while loading the operating system and drivers, measurements of the system are taken, and a reproducible hash value is saved in this register.

The HSM use-flags are a low-level access control for key usage, which we employ for this protocol. Depending on the use of secure storage, different additional measures are required:

- Use-flags of the key are bound to a securely booted platform via ECR.
- Use-flags of the key are bound to a specific execution context and bootstrap phase via ECR
- Use-flags of the key structure are bound to an application-supplied credential (for application-data and user-data),
- Secondary key material (explained below) is protected by standard operating system measures such as file permissions.

B.3.2 Native Encryption

This method uses the HSM directly. Using the HSM processor may limit the overall throughput for other applications utilizing the HSM. A symmetric key is

generated with corresponding use-flags to enforce low level access control (see above). It is then employed using the HSM API via the framework, notably the functions `_encrypt_init()`, `_update()`, and `_finish()`. The output of the update function is sequentially written into a file on the local filesystem.

B.3.3 Secondary Encryption

A way to limit the use of resources of the HSM is to use a secondary key that is stored in non-volatile memory. This secondary key is stored using native HSM encryption as explained above. In particular this means that only the secondary key is directly encrypted and decrypted by the HSM. The use of this key must be bound to a certain execution context with a specific ECR configuration, so that platform integrity is ensured. In Figure B.4 a schematic illustrating secondary encryption is shown. The actual encryption key k_2 is used to encrypt data d and is secured by a key within the HSM (k_1). Data stored on the filesystem (k_2 and d) are protected by encryption, while k_1 is stored by the HSM internally, guarded by low level use-flags. The schematic shows how the secure storage API requests data (step 1), which itself uses the HSM API to retrieve and decrypt the corresponding secondary key (steps 2–4) and finally decrypts data (5) and delivers it to the requesting application (steps 6–7).

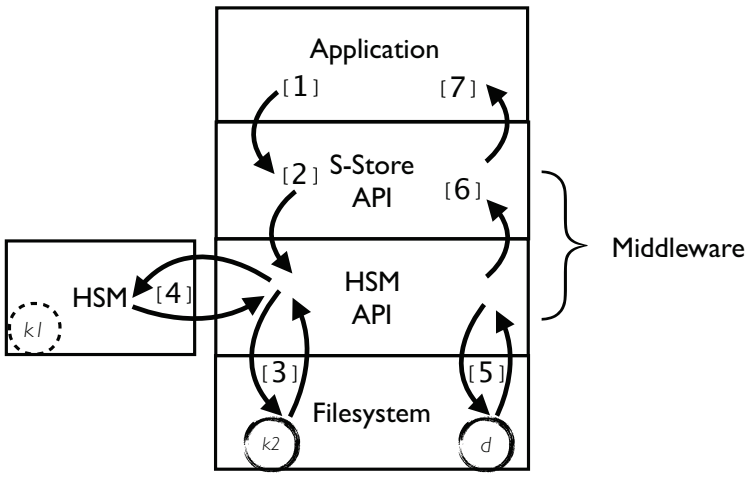


Figure B.4: Secure Storage in Software.

B.4 Intrusion Detection Sensors

The intrusion detection framework described in Section 5.1 was prototypically implemented as part of the EMVY framework. It already provided basic functionality for receiving `SecurityEventNotices` from other modules of the framework.

The sensors that can be part of the local EMVY instance, or stand-alone instances distributed over the vehicle network send probe data. This data is consumed by evaluator functions at the master node. They are subscribed to specific sensors, e.g., one evaluator function may be subscribed to several network detectors.

Testing our approach in the scope of real world vehicle attacks is rather difficult. In contrast to normal computer systems, there are no widely known attacks or signature databases, or standardized tests. Looking into the PC domain, even there no tests have been standardized, although being discussed for years [AMT08]. Due to that, malware detection may be very efficient on a small specialized test set, but loses much of its applicability on a broader set, as found in recent work comparing system-call based anomaly detection [CLB+12].

Due to the fact that we cannot show the effectiveness on a real on-board network, we have implemented a small sensor based on the amount of TCP connections being opened. The data is forwarded to the master, where a filter will evaluate the absolute and the relative number of connections. An action is triggered when either i) an absolute threshold is exceeded or ii) a relative threshold of newly opened or closed connections is exceeded.

The Sensor The sensor that sends in order to obtain the network connection number and type from Unix-compatible systems that served as proof of concept (tested on Linux, MacOS X, cygwin-windows). The number of currently open TCP connections is serialized and sent upstream to the evaluator function. A sample is taken every second. We used very a basic approach to receive the number of open connections: the `netstat` utility provides informations about local connections, which we summed up using the `grep` and `wc` utilities and read that into a local variable. This code provided new probe data every second.

```
fp = popen("netstat -t -n | grep 'EST' | wc -l", "r");
fgets(inbuf, sizeof(inbuf)-1, fp)
int conns = atoi(inbuf);
```

The Filters We have crafted the filter in a way that it can be applied to multiple of types of automotive sensor data.

A large number of models for malfunction or anomaly detection rely on time series. While some models can be as complex as correlating data streams or event streams, a very simple evaluation of signals is, in most cases, sufficient.

In the automotive domain, typically a timed window is applied to a series of data is a common approach in order to detect malfunction of vehicle components. For instance, such detections rely on conditions like "The gasoline level of the tank shall not drop by more than 20% within half an hour". If such a condition is met, the according ECU sets a so-called Diagnostic Trouble Code (DTC) that is stored along with debug information. This indicates the potential fault to mechanics at the next workshop visit.

Consequently, we have designed a generic derivate filter as well as a classic limiting event filter: the `LimitEventFilter` can be parameterized with minimum and maximum allowed values (also commonly used for vehicle diagnosis: for instance, cooling-water temperature shall not exceed 120° C.). To allow a more fine-grained decision, the `FirstDerivateFilter` uses the actual differences between measurements and can be parameterized with a maximum (negative or positive) gradient (e.g., if the temperature rises by more than 20° C in one minute).

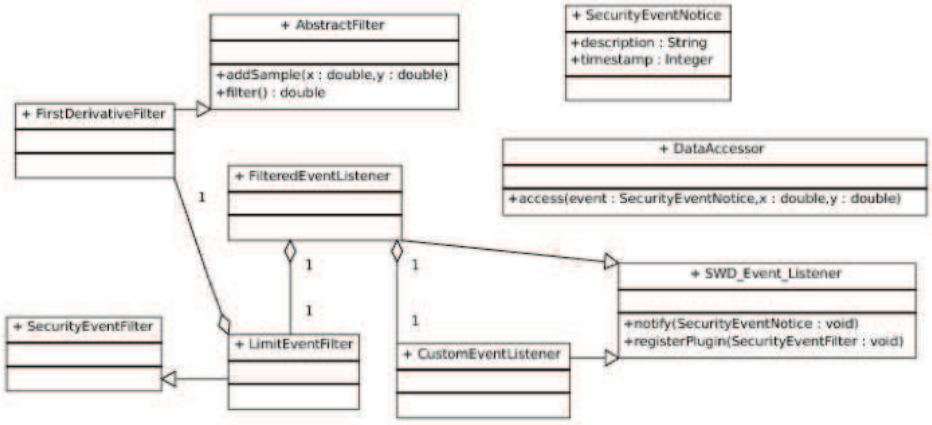


Figure B.5: UML Architecture of Generic Filter.

B.4.0.1 Experimenta Data

We have recorded data with a Probe and the FirstDerivate filter in place. You can see the number of open TCP connections on the computer in Figure B.6. The corresponding derivate plot is shown in Figure B.7. You can clearly see how an absolute and a derivate filter can be engaged at certain thresholds, e.g., the

amount of total open connections could be limited at 25 (LimitEventFilter) or at 1.5 additional connections per second (FirstDerivateFilter).

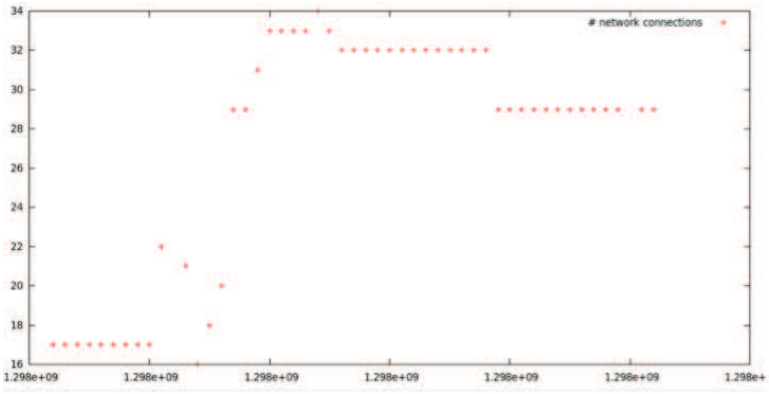


Figure B.6: Absolute amount of open network (TCP) connections at a time. Samples are taken every second.

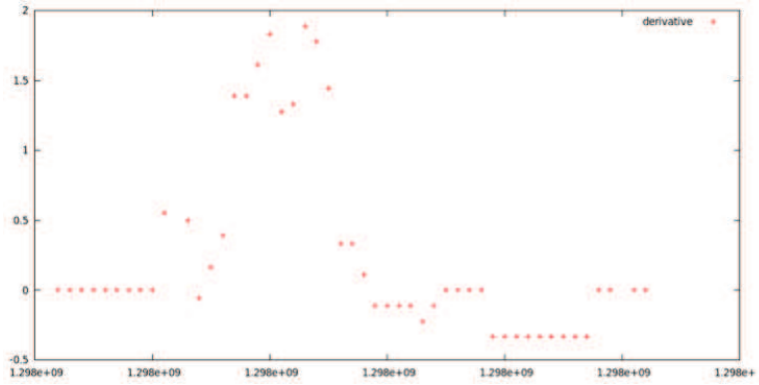


Figure B.7: Relative amount of opening and closing network (TCP) connections per second (first derivative). The time between adjacent data points is one second.

B.5 CAN Ethernet Gateway

The CAN Ethernet Gateway is based on an example application that was part of the SocketCAN 'trunk' repository's experimental applications. The existing application used a TCP-IP listening socket in order to relay input data (given in ASCII hex) to the CAN interface.

The Gateway uses TCP/IP or UDP connections on the Ethernet side, and the ISO-TP protocol on the CAN side. For communicating with an ECU on the CAN bus, the ISO-TP protocol employs a pair of CAN identifiers that are used for ingoing and outgoing traffic.

The gateway application was complemented with

i) an active connection mode (option `-c`),

ii) a raw data relay mode for 4095-byte packets,

iii) a UDP/multicast mode, in which group communication is forwarded to individual CAN nodes.

In the following listing B.5, we show the help screen of the gateway application that is implemented in user-space using SocketCAN and IP sockets in a `select()` loop.

We contributed our program code to the SocketCAN open source community.

Listing B.5: Usage screen of the CAN Ethernet gateway application.

```
Usage: isotp-gateway -l <port>    -s <can_id> -d <can_id> [options] <CAN interface>
Or   : isotp-gateway -c <IP:port> -s <can_id> -d <can_id> [options] <CAN interface>
Options: (* = mandatory)

ip adressing:
 *       -l <port>    (local port for the server)
 *       -c <IP:port> (remote IP and port to connect to)
         -u           (UDP mode, default is TCP)
         -M           (multicast mode, combine with -c <IP:port>)
         -A           (ASCII on IP socket, input expected as HEX strings"<AABB>")
         -R           (default is RAW mode: binary data is relayed)

isotp adressing:
 *       -s <can_id>  (source can_id. Use 8 digits for extended IDs)
 *       -d <can_id>  (destination can_id. Use 8 digits for extended IDs)
         -x <addr>    (extended addressing mode)

padding:
         -p <byte>    (set and enable tx padding byte)
         -r <byte>    (set and enable rx padding byte)
         -P <mode>    (check padding in SF/CF. (l)ength (c)ontent (a)ll)

rx path: (config, which is sent to the sender / data source)
         -b <bs>      (blocksize. 0 = off)
         -m <val>     (STmin in ms/ns. See spec.)
         -w <num>     (max. wait frame transmissions)

tx path: (config, which changes local tx settings)
         -t <time ns> (transmit time in nanosecs)

All values except for '-l', '-c', '-m', and '-t' are expected in hexadecimal values.
```

B.6 Active Brake Prototype Demonstrator

As part of the Car2Car Forum 2011, the final EVITA workshop was conducted with two demonstrator vehicles from BMW. As the work of this thesis was contributed to this EU project, specifically the communication protocols and their integration with the vehicle framework and the HSMs, were part of the demonstrator prototype.

A similar setup with one vehicle was presented at a joint EURECOM/BMW press event in September 2011 and on the 9th conference "embedded security in cars" (escar) on November 8–9 in Dresden. The ECUs of the sending vehicle were installed on a desktop and the brake signal was manually triggered.

In the following, we show two pictures of the demonstrator vehicles in Figures B.8 and B.9. In Figure B.10 the sim^{TD} CCU can be seen on top of the EVITA HSM case, which hosts four prototype boards.

Figure B.8: EVITA Active Brake on the Road. A red vehicle appears in the dashboard. Picture courtesy of BMW Press [BP10a].

Figure B.9: Active Brake Prototype Demonstrator at Car2Car Forum 2012: The sending vehicle is shown in the background (brake lights engaged). The screen in the background shows details of the key distribution protocol and secure communication: console outputs of brake sensor, key master and brake unit are revealed. In the foreground, the receiving vehicle shows a warning message to the driver: A white vehicle with brake lights appears on the instrument cluster's display.

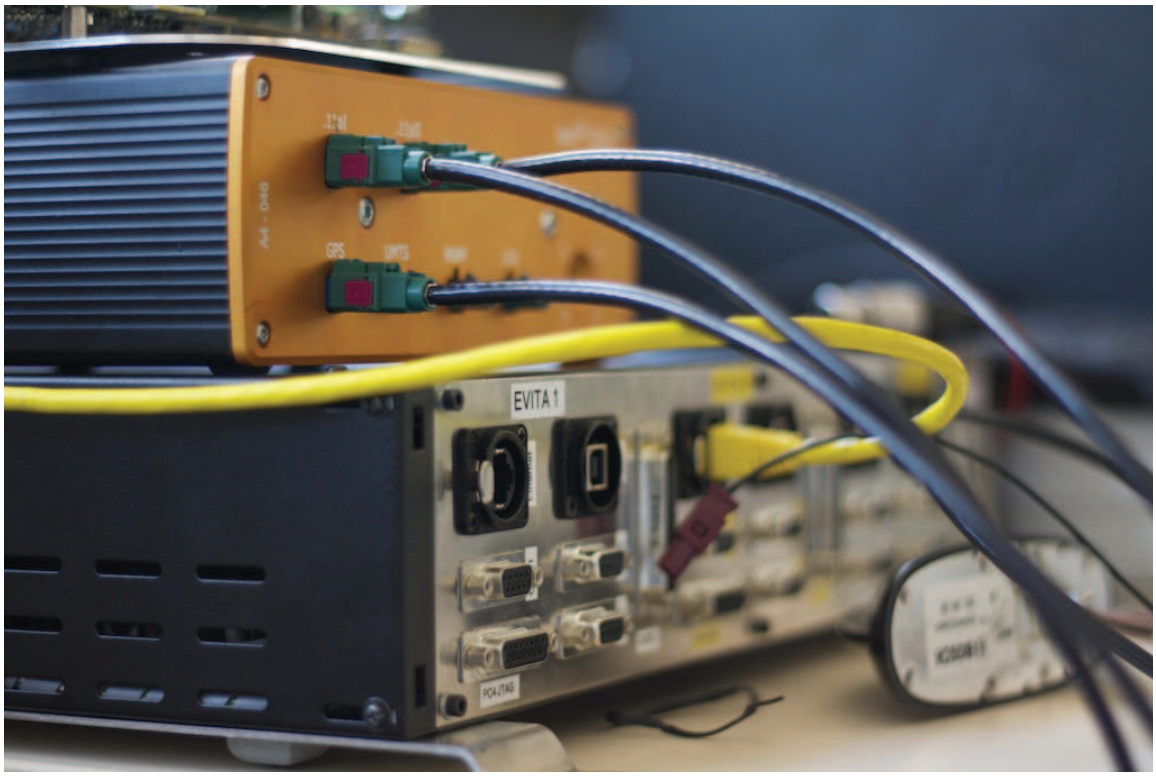


Figure B.10: Picture of the sim^{TD} CCU and EVITA HSM case. The case holds four FPGA boards and is deployed inside the prototype vehicle.

Appendix C

Glossary

C.1 Acronyms and Abbreviations

Both the vehicle industry and the computer industry use a large amount of acronyms and abbreviations, which an outsider does not necessarily know. Many of these acronyms are used in this thesis. We have deliberately not included those, which are commonly known in computer science, but focused on those, that are specific to the vehicle domain. This section compiles a list of the acronyms used throughout the thesis. Every acronym is expanded within the thesis at first occurrence. The electronic form of the thesis allows the reader to navigate directly to the explanation by using pdf hyperlinks.

A/C Air Conditioning

ASIC Application-Specific Integrated Circuit

ASLR Address Space Layout Randomization

CAM Cooperative Awareness Message

CAN Controller Area Network

CCU Communication Control Unit, the communication ECU used for Car2X communication.

CCM Communication Control Module, an EMVY component

Car2X Car-to-Car and Car-to-Infrastructure communication, sometimes V2X for Vehicle-to-X communication.

CGW Central Gateway, the central network component of a vehicle.

DENM Distributed Environment Notification Message

DIFT Dynamic Information Flow Tracking

DLNA Digital Living Network Alliance

DoS Denial of Service, a type of attack that hinders a service or device from working.

DSRC Dedicated Short Range Channel, used to refer to the 5.9 GHz channel for Car2X communication.

DTC Diagnostic Trouble Code

ECC Elliptic Curve Cryptography

ECR ECU Configuration Register, similar to the TPM's trusted configuration registers.

ECU Electronic Control Unit

ESS Emergency Stop Signal

ETSI European Telecommunications Standards Institute

EMVY Embedded Vehicular securitY, a vehicular security software framework from BMW, which was used for most prototype implementations.

EVITA E-Safety Vehicle Intrusion proTected Applications, the FP7 European Project in which this thesis was done.

FOT Field Operational Test

FOTA Firmware Over The Air

FPGA Field Programmable Gate Array

HIS Hersteller Initiative Software

HMI Human Machine Interface

HSM Hardware Security Module

HU Head Unit, the central infotainment ECU.

IC Instrument Cluster, the ECU that includes driver displays, e.g., the speedometer.

IDK Identity Key

IDS Intrusion Detection System

IP Internet Protocol

ITS Intelligent Transport Systems

KM Key Master, the central unit for Key Distribution

KMM Key Management Module, an EMVY component

LIN Local Interconnect Network

MAC Message Authentication Code

MCC Model Carrying Code

MOST Multimedia Oriented Systems Transport

MSC Message Sequence Chart

NFC Near Field Communication

OBD On-Board Diagnostics Socket

OBU On-Board Unit

OEM Original Equipment Manufacturer, the vehicle manufacturer.

PCC Proof Carrying Code

RFID Radio-Frequency Identification

PKI Public Key Infrastructure

ROP Return Oriented Programming

RDS Radio Data System

RPC Remote Procedure Call

RSU Road Side Unit

SHE Secure Hardware Extensions, an HSM for automotive use.

SPI Serial Peripheral Interface, a serial bus used in embedded systems.

TESLA Timed Efficient Stream Loss-Tolerant Authentication

TMC Traffic Message Channel

TPM Trusted Platform Module, an HSM established for Personal Computers

UPnP Universal Plug and Play

VANET Vehicular Adhoc NETwork, the adhoc network created by Car2X communication.

VFB Virtual Function Bus

VIN Vehicle Identification Number

V2X See Car2X

WiFi Wireless Ethernet brand name, referring to the IEEE 802.11 family of standards.

WEP Wireless Equivalent Privacy, the original encryption method used for wireless ethernet networks (IEEE 802.11b)

List of Figures

1.1 Vehicle Attack Contexts 5

2.1 On-Board Networks 10
2.2 Electronic Brake Pedal with Redundant Sensors 12
2.3 Car2X Example Use Cases 17
2.4 AUTOSAR Development Paradigm 19
2.5 Partitioning with Proof Carrying Code 23
2.6 Code Lifetime and Possible Security Measures 25
2.7 Taint Propagation in TaintDroid 29
2.8 Classification of Techniques 31
2.9 Enforcement of Security Policies: The Apple AppStore 33
2.10 Timeseries of Apple iDevice Jailbreaks 34
2.11 Cost and Complexity of Security Approaches 35

3.1 Active Brake Scenario with Two Vehicles 40
3.2 Active Brake Scenario - Message Sequence Chart 41
3.3 Active Brake Scenario: Components 42
3.4 Music Player Scenario 43
3.5 Music Player Scenario Sequence Chart 44
3.6 Driver Adaptation Scenario 44
3.7 Driver Adaptation: Message Sequence Chart 45
3.8 EVITA Software Architecture 50
3.9 HSM Hardware Architecture 52

4.1 Key Distribution Deployement 58

4.2 Key Distribution 59
4.3 Phases of Key Distribution 61
4.4 Key Distribution with Domain Partitioned Keyspace 63
4.5 Keying at Maintenance - Online Case 65
4.6 Keying at Maintenance - Offline Case 66
4.7 CAN Transport Protocol 70
4.8 Histogram of Risk Level for Automotive Use-Cases 71

5.1 IDS Architecture 77
5.2 IDS Deployment 79
5.3 Tainting for Vehicle Architecture 83
5.4 Distributed DFT Architecture 85
5.5 Deployment of Multiple Applications on Head Unit 87
5.6 Propagation of Taint in Scenario III 87
5.7 Marshalling of Taint Data at Network Level 90
5.8 Runtime Checks By Controlling Key Usage 91
5.9 MSC: Timed behavior of key usage 92

6.1 CAN frames and payload per second 99
6.2 CAN identifier distribution 100
6.3 Simulation with TTool 102
6.4 Simulink TrueTime Setup 103
6.5 CAN Signal Latency 104
6.6 Performance Comparison for Different Taint Engines 108
6.7 CAN Ethernet Gateway 110

A.1 Contextes d'attaque de voitures 124
A.2 Car2X Example Use Cases 129
A.3 Paradigme de Developpement AUTOSAR 134
A.4 Propagation des marquages dans TaintDroid 137
A.5 Architecture de HSM prototype 143
A.6 Scénario Freinage Actif 145

A.7 Scenario Écoute Musique 146
A.8 Scenario Adaptation au Conducteur 147
A.9 Architecture de Gestion de Clés 151
A.10 Les Phases de Distribution des Clés 152
A.11 Architecture Distribué DFT 156

B.1 Per-Byte HSM Performance of CMAC 164
B.2 HSM Runtime Overhead Breakdown 165
B.3 Detailed Group Communication Establishment 170
B.4 Secure Storage in Software 172
B.5 UML Architecture of Generic Filter. 174
B.6 IDS: Open Network Connections (absolute) 175
B.7 IDS: Open Network Connections (relative) 175
B.8 Picture: Active Brake on the Road 178
B.9 Picture: Active Brake Prototype 179
B.10 Picture: The simTD CCU and EVITA HSM 180

Listings

5.1 Application Code Example for Taint Tag Propagation 88
B.1 Application Code at Client Side (Sender). 166
B.2 Application Code at Client Side (Receiver). 167
B.3 Framework Program Code (Application/Client Side). 167
B.4 Framework Program Code (Server Side). 168
B.5 Usage screen of the CAN Ethernet gateway application. 177

List of Tables

2.1 Steps for Policy Enforcement 33

3.1 ECU Abbreviations in Scenario I 40
3.2 Comparison between HSM Solutions 54

4.1 Expected Time to Collide Truncated MAC 72

6.1 Frame and Payload per Second on Body CAN-Bus 98
6.2 Latency of Secured CAN-TP Frames 105
6.3 Instrumented Runtimes of mpg123 107

B.1 HSM Processing Times 165
B.2 Relative Overhead of HSM Processing 165

List of Publications 2009—2012

Conference and Journal Publications

1. H. Schweppe, Y. Roudier, *"Security and Privacy for In-Vehicle Networks"*, IEEE VCSC, 1st International Workshop on Vehicular Communications, Sensing, and Computing, 18 June 2012, Seoul, Republic of Korea.

2. H. Schweppe, B. Weyl, Y. Roudier, M-S. Idrees, T. Gendrullis, M. Wolf, *"Securing car2X applications with effective hardware-software co-design for vehicular on-board networks"*, 27th VDI Conference on Automotive Security, appeared in VDI-Bericht 2131, pp 45–57, October 11–12, 2011, Berlin, Germany.

3. H. Schweppe, Y. Roudier, B. Weyl, L. Apvrille, D. Scheuermann, *"Car2X communication: Securing the last meter"*, WIVEC 2011, 4th IEEE International Symposium on Wireless Vehicular Communications, 5–6 September 2011, San Francisco, USA.

4. M-S. Idrees, H. Schweppe, Y. Roudier, M. Wolf, D. Scheuermann, and O. Henniger *"Secure Automotive On-Board Protocols : A Case of Over-the-Air Firmware Updates"*, in Nets4Cars, 3rd International Workshop on Communication Technologies for Vehicles, Oberpfaffenhofen Germany, 22–24 March 2011, appeared in Springer LNCS 6596/2011, pp 224–238.

5. L. Apvrille, R. El Khayari, O. Henniger, Y. Roudier, H. Schweppe, H. Seudié, B. Wey, M. Wolf, *"Secure automotive on-board electronics network architecture"*, FISITA 2010, 32nd World Automotive Congress, 30 May–4 June 2010, Budapest, Hungary.

6. H. Schweppe, A. Zimmermann, and D. Grill, *"Flexible On-Board Stream Processing for Automotive Sensor Data"*, IEEE Transactions on Industrial Informatics, Vol. 6, No. 1, pp 81–92, February 2010, DOI 2009.10.1109/TII.2009.2037145, ISSN: 1551-3203. (publication of previous research)

Invited Presentations, Poster Presentations, and Short Papers

1. H. Schweppe, *"Securing Car2X Applications with effective Hardware-Software Co-Design for Vehicular On-Board Networks"*, invited talk at Volkswagen AutoUni, 30 Nov. 2011, Wolfsburg, Germany.

2. H. Schweppe, *"The EVITA project"*, Posters and Vehicle Demonstrator at 5th Car2Car CC Forum, 24–25 Nov. 2011, Erlensee, Germany.

3. H. Schweppe, *"Secure on-board protocols"*, Presentation at Final EVITA Workshop on Security of Automotive On-Board Networks, 23 Nov. 2011, Erlensee, Germany

4. B. Weyl, H. Schweppe, *"The EVITA project: securing the networked vehicle"*, invited Keynote Presentation at 9th escar conference on embedded security in cars (escar), 9–10 Nov. 2011, Dresden, Germany.

5. H. Schweppe, *"EVITA: Secure On-Board Protocols"*, Joint BMW/EURECOM Press Workshop, Posters and Vehicle Demonstrator, September 30 2011, Sophia-Antipolis, France.

6. Presented for: G. Pedroza, S. Idrees, L. Apvrille, Y. Roudier, *"A formal methodology applied to secure over-the-air automotive applications"*, VTC-Fall2011, IEEE 74th Vehicular Technology Conference -VTC Fall 2011, 5-8 September 2011, San Francisco, USA

7. H. Schweppe, *"Security Requirements for Intersection Collision Warning"*, Workshop Presentation at 4th Car2Car CC Forum, November 23–24 2010, Paris, France.

8. H. Schweppe, *"The EVITA project, an overview"*, Poster at 4th Car2Car CC Forum, Nov. 23–24 2010, Paris, France.

9. H. Schweppe, *"Secure on-board protocols"*, CAST-Workshop on Mobile Security for Intelligent Cars, Presentation and Short Paper, July 1, 2010, Darmstadt, Germany

10. H. Schweppe, Y. Roudier, *"Security issues in vehicular systems: threats* emerging solutions and standards", SAR-SSI 2010, 5th Conference Conference on Security in Network Architectures and Information Systems, State of the Art Presentation and Short Paper, 18–21 May 2010, Menton, France.

Reviews

1. IEEE Vehicular Networking Conference, 2012
2. IEEE Wireless On-demand Network Systems and Services, 2012
3. IEEE Vehicular Technology Conference, Fall 2011
4. IEEE Security and Privacy, 2011
5. IEEE Conference on Pervasive Computing and Communications, 2011
6. IEEE Journal on Selected Areas in Communications, Special Issue Vehicular Communications and Networks, 2010
7. IEEE Security and Privacy, 2010

Teaching

1. Guest lecture as part of E. Kirda's course *"Secure Programming II"*: $Security\ in\ Modern\ Vehicles$, EURECOM, January 2010.
2. Guest lecture as part of Y. Roudier's course *"Distributed Systems and Middleware"*: $DLNA\ and\ UPnP/AV$, EURECOM, May 2012.
3. Guidance of exercises and laboratories, part of Y. Roudier's course *"Distributed Systems and Middleware'*, EURECOM, 2009, 2010, 2011, 2012.
4. Student semester project: *"Distributed and Embedded Intrusion Detection for Automotive Systems"*, EURECOM, Rahul Sinha, fall 2009 to summer 2010.
5. Student semester project: *"Security Middleware for Automotive Embedded Systems"*, EURECOM, Richard Gross and Jonas Zaddach, fall 2010 to spring 2011.
6. Student semester project: *"Simulating In-Vehicular Networking on the CAN bus"*, EURECOM, Ning Qi, fall 2011 to spring 2012, co-supervised with Jérôme Härri.

Bibliography

[AEKH+10] Ludovic Apvrille, Rachid El Khayari, Olaf Henniger, Yves Roudier, Hendrik Schweppe, Herve Seudié, Benjamin Weyl, and Marko Wolf, *Secure automotive on-board electronics network architecture*, FISITA'10, World Automotive Congress, 30 May-4 June, 2010, Budapest, Hungary, 2010. (Cited on page 49.)

[AMAB+06] L. Apvrille, W. Muhammad, R. Ameur-Boulifa, S. Coudert, and R. Pacalet, *A UML-based environment for system design space exploration*, ICECS '06. 13th IEEE International Conference on Electronics, Circuits and Systems. (Nice, France), Dec 2006, pp. 1272 –1275. (Cited on page 101.)

[AMT08] AMTSO, *Anti-Malware Testing Standards Organization*, 2008. (Cited on page 173.)

[And02] Ross Anderson, *Why information security is hard - an economic perspective*, Seventeenth Annual Computer Security Applications Conference, IEEE, June 2002, pp. 358–365 (English). (Cited on pages 36 and 138.)

[APH08] Elvira Albert, Germán Puebla, and Manuel Hermenegildo, *Abstraction-carrying code: a model for mobile code safety*, New Generation Computing **26** (2008), 171–204. (Cited on page 24.)

[App12] Apple Inc., *ios security*, Tech. report, Apple Inc., Cupertino, May 2012. (Cited on pages 34 and 75.)

[AUT11] AUTOSAR, *Specification of crypto abstraction library v1.2.0 r4.0 rev. 3*, online, http://www.autosar.org/, 12 2011. (Cited on pages 18 and 135.)

[BB07] Andrea Barisani and Daniele Bianco, *Hijacking RDS-TMC traffic information signals*, The Phrack Magazine **64** (2007), no. 5. (Cited on page 12.)

[Be09] Hagai Bar-el, *Intra-Vehicle Information Security Framework*, escar, embedded security in cars, 2009. (Cited on pages 15, 73, and 133.)

[BFWS10] Stefan Bunzel, S Fürst, J Wagenhuber, and F Stappert, *Safety and security related features in AUTOSAR*, Automotive - Safety & Security 2010, 2010. (Cited on pages 18, 19, 115, 134, 135, and 159.)

[Bis04] Matt Bishop, *Introduction to computer security*, Addison-Wesley Professional, 2004. (Cited on page 77.)

[BMKK06] Ulrich Bayer, Andreas Moser, Christopher Kruegel, and Engin Kirda, *Dynamic Analysis of Malicious Code*, Journal in Computer Virology **2** (2006), no. 1, 67–77. (Cited on page 27.)

[BP10a] BMW-Pressestelle, *BMW Group PressClub*, 2010. (Cited on pages 10 and 178.)

[BP10b] _____, *Mehr Fahrspass durch Vernetzung: MINI Connected.*, 2010. (Cited on pages 3 and 10.)

[Bru10] Robert Bruckmeier, *Ethernet for Automotive Applications*, Freescale Technology Forum (Orlando), 2010, pp. 1–20. (Cited on pages 13, 115, and 159.)

[BSM+09] Norbert Bißmeyer, Hagen Stübing, Manuel Mattheß, Jan Peter Stotz, Julian Schütte, Matthias Gerlach, and Florian Friederici, *simTD Security Architecture: Deployment of a Security and Privacy Architecture in Field Operational Tests*, 7th ESCAR Embedded Security in Cars Conference, Düsseldorf (2009) (en). (Cited on pages 15, 55, and 131.)

[CC11] Jason Croft and Matthew Caesar, *Towards Practical Avoidance of Information Leakage in Enterprise Networks*, 6th USENIX Workshop on Hot Topics in Security (San Francisco), 2011. (Cited on page 94.)

[CGL+11] George Coker, Joshua Guttman, Peter Loscocco, Amy Herzog, Jonathan Millen, Brian O'Hanlon, John Ramsdell, Ariel Segall, Justin Sheehy, and Brian Sniffen, *Principles of remote attestation*, International Journal of Information Security - Special Issue: 10th International Conference on Information and Communications Security (ICICS) **10** (2011), no. 2, 63–81. (Cited on pages 6 and 125.)

[Che09] Winnie Wing-Yee Cheng, *Information Flow for Secure Distributed Applications*, Phd thesis, MIT, 2009. (Cited on pages 28, 94, and 137.)

[CHL+03] By Anton Cervin, Dan Henriksson, Bo Lincoln, Johan Eker, and Karl-erik Å rzén, *Analysis and Simulation of Timing*, IEEE Control Systems Magazine (2003), no. June, 16–30. (Cited on page 102.)

[CLB+12] Davide Canali, Andrea Lanzi, Davide Balzarotti, Christopher Kruegel, Mihai Christodorescu, and Engin Kirda, *A quantitative study of accuracy in system call-based malware detection categories and subject descriptors*, Proceedings of the 2012 International Symposium on Software Testing and Analysis (ACM ISSTA 2012), July 15-20 2012. (Cited on page 173.)

[CLO07] James Clause, Wanchun Li, and Alessandro Orso, *Dytan: a generic dynamic taint analysis framework*, Symposium on Software testing and analysis, 2007, pp. 196–206. (Cited on pages 84, 93, 106, and 156.)

[CMK+11] Stephen Checkoway, Damon McCoy, Brian Kantor, Danny Anderson, Hovav Shacham, Stefan Savage, Karl Koscher, Alexei Czeskis, Franziska Roesner, and Tadayoshi Kohno, *Comprehensive experimental analyses of automotive attack surfaces*, Proceedings of the 20th USENIX conference on Security, 2011. (Cited on pages 43, 46, 75, 87, 94, 117, 146, 153, and 162.)

[CRH05] Miguel Leon Chavez, Carlos Hernandez Rosete, and Francisco Rodriguez Henriquez, *Achieving confidentiality security service for can*, Proceedings of the 15th International Conference on Electronics, Communications and Computers (Washington, DC, USA), CONIELECOMP '05, IEEE Computer Society, 2005, pp. 166–170. (Cited on pages 15, 73, and 133.)

[Dwo05] Morris Dworkin, *Cipher modes of operation; the cmac mode for authentication.*, NIST Special Publication 800-38b. National Institute of Standards and Technology (NIST), May 2005. (Cited on page 69.)

[EB06] Thomas Eymann and Michael Busse, *Deliverable D1.2-12 Security and Firewall concepts for gateways*, Tech. report, EASIS-Project, 2006. (Cited on pages 14, 73, and 132.)

[EGC+10] William Enck, Peter Gilbert, Byuong-Gon Chun, Landon P. Cox, Laejeon Jung, Patrick McDaniel, and Anmol N. Sheth, *TaintDroid: An information-flow tracking system for realtime pri-*

vacy monitoring on smartphones, Operating Systems Design and Implementation, USENIX 2010, 2010. (Cited on pages 24, 28, 29, 81, 116, 136, 137, and 160.)

[EKKV11] Manuel Egele, Christopher Kruegel, Engin Kirda, and Giovanni Vigna, *PiOS: Detecting privacy leaks in iOS applications*, Network and Distributed System Security Symposium (NDSS), The Internet Society, 2011. (Cited on page 81.)

[EKM+08] Thomas Eisenbarth, Timo Kasper, Amir Moradi, Christof Paar, Mahmoud Salmasizadeh, and Mohammad T Manzuri Shalmani, *On the Power of Power Analysis in the Real World : A Complete Break of the KeeLoq Code Hopping Scheme*, CRYPTO, 2008. (Cited on page 13.)

[EKS+10] Andrey Ermolinskiy, Sachin Katti, Scott Shenker, Lisa L Fowler, and Murphy McCauley, *Towards practical taint tracking*, Tech. Report UCB/EECS-2010-92, EECS Department, University of California, Berkeley, Jun 2010. (Cited on pages 84 and 155.)

[ES00] C. Ellison and Bruce Schneier, *Ten risks of PKI: What you're not being told about public key infrastructure*, Computer Security Journal **16** (2000), no. 1, 1–7. (Cited on page 22.)

[ESKK08] Manuel Egele, Theodoor Scholte, Engin Kirda, and Christopher Kruegel, *A survey on automated dynamic malware-analysis techniques and tools*, ACM Computing Surveys **44** (2008), no. 2, 6:1–6:42. (Cited on pages 27, 77, 116, and 161.)

[Eur09a] European Commission, The, *eCall – saving lives through in-vehicle communication technology*, online, factsheet 49, August 2009. (Cited on pages 10, 16, and 132.)

[Eur09b] _____, *M/453 EN standardisation mandate addressed to CEN, CENELEC and ETSI in the field of information and communication technologies to support the interoperability of co-operative systems for intelligent transport in the european community*, October 2009. (Cited on pages 16, 55, 117, 132, 148, and 161.)

[EVI11] EVITA, *The EVITA project*, http://evita-project.org/, August 2008–2011. (Cited on page 49.)

[FDC11] A. Francillon, Boris Danev, and Srdjan Capkun, *Relay Attacks on Passive Keyless Entry and Start Systems in Modern Cars*, 18th Annual Network & Distributed System Security Symposium, Cryptology ePrint Archive, Report 2010/332, 2010, 2011. (Cited on pages 13 and 73.)

[Fed02] Federal Information Processing Standards, *The key-hash message authentication code (HMAC)*, FIPS Publication 198, Federal Information Processing Standards, March 2002. (Cited on page 69.)

[FFH+04] Andreas Festag, Holger Füßler, Hannes Hartenstein, Amardeo Sarma, and Ralf Schmitz, *Fleetnet: Bringing car-to-car communication into the real world*, Proceedings of 11th World Congress on ITS, Nagoya, Japan, 2004. (Cited on pages 14, 16, 129, and 131.)

[Fle02] Eric (Boeing) Fleischman, *Code Signing*, The Internet Protocol Journal **5** (2002), no. 1, 14–26. (Cited on page 22.)

[FPC09] Aurélien Francillon, Daniele Perito, and Claude Castelluccia, *Defending embedded systems against control flow attacks*, Proceedings of the first ACM workshop on Secure execution of untrusted code (New York, NY, USA), SecuCode '09, ACM, 2009, pp. 19–26. (Cited on page 25.)

[G11] Stefan Götz, *G5A Automotive Security – Lastanforderung an die Kryptoeinheit (in German)*, VDI 27. Gemeinschaftstagung on Automotive Security (2011), 71–80. (Cited on pages 53 and 163.)

[GBW07] Jinhua Guo, John Baugh, and Shengquan Wang, *A Group Signature Based Secure and Privacy-Preserving Vehicular Communication Framework*, 2007 Mobile Networking for Vehicular Environments (2007), 103–108. (Cited on pages 6, 73, and 125.)

[GEN09] GENIVI Alliance, *Genivi fact sheet*, 2009. (Cited on pages 4, 14, and 122.)

[GFL+07] Matthias Gerlach, Andreas Festag, Tim Leinmüller, Gabriele Goldacker, and Charles Harsch, *Security architecture for vehicular communication*, Fourth International Workshop on Intelligent Transportation (WIT2007), 2007. (Cited on pages 6, 14, 73, 125, and 132.)

[GGZ+08] R. Gupta, N. Gupta, Xiangyu Zhang, D. Jeffrey, V. Nagarajan, S. Tallam, and Chen Tian, *Scalable dynamic information flow tracking and its applications*, Parallel and Distributed Processing, 2008. IPDPS 2008. IEEE International Symposium on, 14-18 2008, pp. 1 –5. (Cited on page 26.)

[GHM+10] Michael Glass, Daniel Herrscher, Herbert Meier, Martin Plastowski, and Peter Schoo, *SEIS - security in embedded ip-based systems*, ATZ elektronik worldwide 1/2010, p. 36, 01 2010. (Cited on pages 10, 15, 115, 133, and 159.)

[GHR+09] André Groll, Jan Holle, Christoph Ruland, Marko Wolf, Thomas Wollinger, and Frank Zweers, *OVERSEE a secure and open communication and runtime platform for innovative automotive applications*, 7th ESCAR Embedded Security in Cars Conference, Düsseldorf (2009) (en). (Cited on pages 15 and 132.)

[Gol10] Dieter Gollmann, *Computer security*, Wiley Interdisciplinary Reviews: Computational Statistics (2010), 544–554. (Cited on page 30.)

[GR09] A. Groll and C. Ruland, *Secure and authentic communication on existing in-vehicle networks*, Intelligent Vehicles Symposium, 2009 IEEE, June 2009, pp. 1093 –1097. (Cited on pages 15, 73, and 133.)

[Har11] Oliver Hartkopp, *Programmierschnittstellen für eingebettete Netzwerke in Mehrbenutzerbetriebssystemen am Beispiel des Controller Area Network*, Ph.D. thesis, OvGU Magdeburg, 2011, p. 238. (Cited on page 109.)

[HFS98] SA Hofmeyr, Stephanie Forrest, and Anil Somayaji, *Intrusion detection using sequences of system calls*, Journal of Computer Security **6** (1998), 151–180. (Cited on page 24.)

[HH08] André Hergenhan and Gernot Heiser, *Operating systems technology for converged ECUs*, 6th Embedded Security in Cars Conference (escar) (Hamburg, Germany), ISITS, Nov 2008. (Cited on page 14.)

[Hig10] Kelly Jackson Higgins, *Smartphone Weather App Builds A Mobile Botnet*, Dark Reading (2010). (Cited on page 24.)

[HIS09] HIS, AK Security, *She: Secure hardware extension version 1.1*, 2009. (Cited on page 53.)

[HKD08a] Tobias Hoppe, Stefan Kiltz, and Jana Dittmann, *Adaptive dynamic reaction to automotive it security incidents using multimedia car environment*, Proceedings of the 2008 The Fourth International Conference on Information Assurance and Security (Washington, DC, USA), IAS '08, IEEE Computer Society, 2008, pp. 295–298. (Cited on pages 116 and 160.)

[HKD08b] ———, *Security threats to automotive can networks — practical examples and selected short-term countermeasures*, SAFECOMP '08: Proceedings of the 27th international conference on Computer Safety, Reliability, and Security (Berlin, Heidelberg), Springer-Verlag, 2008, pp. 235–248. (Cited on page 12.)

[HL06] Yih-chun Hu and Kenneth P Laberteaux, *Strong VANET Security on a Shoestring*, escar, 2006, pp. 1–9. (Cited on pages 6, 73, and 125.)

[HL10] Hannes Hartenstein and Kenneth P. Laberteaux (eds.), *Vanet - vehicular applications and inter-networking technologies*, Intelligent Transport Systems, Wiley, 2010. (Cited on pages 16, 17, 130, and 132.)

[HSV11] Anthony Van Herrewege, Dave Singelee, and Ingrid Verbauwhede, *CANAuth - A Simple , Backward Compatible Broadcast Authentication Protocol for CAN bus*, escar 2011, embedded security in cars, 2011. (Cited on pages 15, 67, 74, 133, and 151.)

[IKD+08] Sebastiaan Indesteege, Nathan Keller, Orr Dunkelman, Eli Biham, and B. Preneel, *A practical attack on KeeLoq*, EUROCRYPT, Springer-Verlag, 2008. (Cited on page 13.)

[ISO00] ISO/IEC IS, *ISO 14230:2000(E) – Road vehicles – Diagnostic systems – Keyword Protocol 2000*, TC 22/SC 3, International Organization for Standardization, Geneva, Switzerland (2000). (Cited on page 14.)

[ISO04] _____, *ISO 15765-2:2004(E) – Road vehicles — Diagnostics on Controller Area Networks (CAN) — Part 2: Network layer services*, TC 22/SC 3, International Organization for Standardization, Geneva, Switzerland (2004). (Cited on pages 67, 68, 70, and 151.)

[ISO05] _____, *ISO 17356 open interface for embedded automotive applications*, TC 22/SC 3, International Organization for Standardization, Geneva, Switzerland (2005). (Cited on pages 18 and 134.)

[ISR+11] Sabir Idrees, Hendrik Schweppe, Yves Roudier, Marko Wolf, Dirk Scheuermann, and Olaf Henniger, *Secure automotive On-Board protocols: A case of over-the-air (OTA) firmware updates*, Nets4Cars-2011 3rd International Workshop on Communication Technologies for Vehicles (Nets4Cars-2011) (Oberpfaffenhofen-Wessling, Munich, Germany), 3 2011. (Cited on page 64.)

[ISS02] R. Isermann, R. Schwarz, and S. Stolzl, *Fault-tolerant drive-by-wire systems*, Control Systems Magazine, IEEE **22** (2002), no. 5, 64–81. (Cited on pages 11, 12, and 128.)

[Jac12] Bernd Jacobi, *VW opener and alarm deactivator*, Multipick-Service tool 70470, manual online at www.multipick.com, 2012. (Cited on pages 79, 115, and 159.)

[JH08] Mike Jones and Eberhard Haug, *Der Einzug von Ethernet in Automobilanwendungen (German)*, Elektronik Praxis (2008), 1–9. (Cited on page 10.)

[Jon10] Mike Jones, *EMI Challenge to Ethernet in the Car*, EE Times Europe (2010), 1–6. (Cited on pages 10, 13, 115, and 159.)

[KAP09] D. Knorreck, L. Apvrille, and R. Pacalet, *Fast simulation techniques for design space exploration*, 47th International Conference Objects, Models, Components, Patterns (Zurich, Switzerland), vol. 33, June 2009, pp. 308–327. (Cited on page 101.)

[KB11] Harry Knechtel and Martin Böhner, *Herausforderung Sicheres Bordnetz (secunet AG and Elektrobit Automotive GmbH, in German)*, VDI 27. Gemeinschaftstagung on Automotive Security (2011), 59–70. (Cited on pages 111, 115, 157, and 159.)

[KBA02] Vladimir Kiriansky, Derek Bruening, and Saman P. Amarasinghe, *Secure execution via program shepherding*, USENIX Security Symposium (Dan Boneh, ed.), USENIX, 2002, pp. 191–206. (Cited on page 26.)

[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*, 31st IEEE Symposium on Security and Privacy, vol. 31, 2010. (Cited on pages 12, 14, 72, 75, 94, 117, 132, 153, and 162.)

[KDK09] Hari Kannan, Michael Dalton, and Christos Kozyrakis, *Decoupling Dynamic Information Flow Tracking with a dedicated coprocessor*, 2009 IEEE/IFIP International Conference on Dependable Systems & Networks, Ieee, June 2009, pp. 105–114. (Cited on pages 28 and 137.)

[Ker83] Auguste Kerckhoffs, *La cryptographie militaire*, Journal des sciences militaires (in French) **IX** (1883), no. 01 and 02, 5–38 and 161–191. (Cited on pages 36 and 139.)

[KFL+09] E. Kelling, M. Friedewald, T. Leimbach, M. Menzel, P. Säger, H. Seudié, and B. Weyl, *Specification and evaluation of e-security relevant use cases*, Tech. Report Deliverable D2.1, EVITA Project, 2009. (Cited on pages 48, 71, and 143.)

[KK99] Oliver Kömmerling and Markus G. Kuhn, *Design principles for tamper-resistant smartcard processors*, Smartcard 99, 1999, pp. 9–20. (Cited on pages 46 and 141.)

[KKB+06] E. Kirda, C. Kruegel, G. Banks, G. Vigna, and R. Kemmerer, *Behavior-based Spyware Detection*, Proceedings of the 15th USENIX Security Symposium (Vancouver, BC, Canada), August 2006. (Cited on page 77.)

[KKCS09] Hyung Chan Kim, Angelos D. Keromytis, Michael Covington, and Ravi Sahita, *Capturing information flow with concatenated dynamic taint analysis*, ARES, IEEE Computer Society, 2009, pp. 355–362. (Cited on pages 28 and 138.)

[KOJ11] P. Kleberger, T. Olovsson, and E. Jonsson, *Security aspects of the in-vehicle network in the connected car*, Intelligent Vehicles Symposium (IV), 2011 IEEE, june 2011, pp. 528 –533. (Cited on pages 16 and 133.)

[Kom10] Paul Kompfner, *The ITS App Store*, ITS International, Intertraffic Amsterdam (Amsterdam), Dailynews, March 2010, pp. 39–41. (Cited on pages 4 and 122.)

[KPB+08] Frank Kargl, Panagiotis Papadimitratos, Levente Buttyan, Michael Müter, Elmar Schoch, Bjorn Wiedersheim, Ta-Vinh Thong, Giorgio Calandriello, Albert Held, Antonio Kung, and Jean-Pierre Hubaux, *Secure vehicular communication systems: implementation, performance, and research challenges*, IEEE Communications Magazine **46** (2008), no. 11, 110–118. (Cited on page 73.)

[KPJK12] Vasileios P. Kermelis, Georgios Portokalidis, Kangkook Jee, and Angelos D. Keromytis, *libdft: Practical Dynamic Data Flow Tracking for Commodity Systems*, VEE'12, SIGPLAN/SIGOPS, 2012. (Cited on pages 28, 84, 93, 106, 107, 138, and 156.)

[Kra04] Jerry Krasner, *Using Elliptic Curve Cryptography (ECC) for Enhanced Embedded Security*, Tech. Report November, Embedded Market Forecasters, 2004. (Cited on page 57.)

[Kun08] Antonio Kung (Ed.), *Security architecture and mechanisms for V2V / V2I*, Tech. Report Deliverable D2.1, Sevecom Project, 2008. (Cited on pages 14, 73, and 132.)

[Law08] Nate Lawson, *Highway to hell: Hacking toll systems*, Blackhat USA (2008). (Cited on page 12.)

[LBH+06] Tim Leinmüller, Levente Buttyan, Jean-Pierre Hubaux, Frank Kargl, Rainer Kroh, Panos Papadimitratos, Maxim Raya, and Elmar Schoch, *SEVECOM - secure vehicle communication*, Proceedings of IST Mobile Summit 2006, 2006. (Cited on pages 14 and 131.)

[LC06a] Lap Chung Lam and Tzi-cker Chiueh, *A general dynamic information flow tracking framework for security applications*, ACSAC '06: Proceedings of the 22nd Annual Computer Security Applications Conference (Washington, DC, USA), IEEE Computer Society, 2006, pp. 463–472. (Cited on pages 27, 135, and 136.)

[LC06b] _____, *A general dynamic information flow tracking framework for security applications*, Proceedings of the 22nd Annual Computer Security Applications Conference (Washington, DC, USA), ACSAC '06, IEEE Computer Society, 2006, pp. 463–472. (Cited on pages 28, 114, 137, and 158.)

[LCM+05] CK Luk, Robert Cohn, Robert Muth, Harish Patil, and Artur Klauser, *Pin: building customized program analysis tools with dynamic instrumentation*, ACM SIGPLAN (2005). (Cited on pages 84, 106, and 156.)

[Lev96] Elias Levy (Aleph One), *Smashing the stack for fun and profit*, The Phrack Magazine **49** (1996), no. 14. (Cited on page 85.)

[Lin09] Marc Lindlbauer, *Towards a Secure Automotive Platform*, Tech. report, secunet, Essen, Germany, 2009. (Cited on pages 15 and 133.)

[LLZ+08] Xiaodong Lin, Rongxing Lu, Chenxi Zhang, Haojin Zhu, Pin-Han Ho, and Xuemin Shen, *Security in vehicular ad hoc networks*, Communications Magazine, IEEE **46** (2008), no. 4, 88 –95. (Cited on pages 16, 132, and 163.)

[LNJ08] U.E. Larson, D.K. Nilsson, and E. Jonsson, *An approach to specification-based attack detection for in-vehicle networks*, Intelligent Vehicles Symposium, 2008 IEEE, june 2008, pp. 220 –225. (Cited on pages 78, 95, 115, and 159.)

[Lut11] Zachary Lutz, *Renault debuts r-link, an in-dash android system with app market*, engadget following press release by Renault at LeWeb'11, December 2011. (Cited on pages 4, 116, 122, and 160.)

[MA11] Michael Müter and Naim Asaj, *Entropy-based anomaly detection for in-vehicle networks*, Intelligent Vehicles Symposium (IV), 2011 IEEE, June 2011, pp. 1110–1115. (Cited on pages 115 and 159.)

[Mai12] Alexander Maier, *Ethernet – the standard for in-car communication*, 2nd Workshop for Ethernet and IP at Automotive Technology Day (Regensburg), September 19–20, 2012. (Cited on page 115.)

[Mar10] John Markoff, *Google cars drive themselves in traffic*, The New York Times, Science, Smarter Than You Think (2010). (Cited on page 11.)

[MBB+08] Michael Montemerlo, Jan Becker, Suhrid Bhat, Hendrik Dahlkamp, Dmitri Dolgov, Scott Ettinger, Dirk Haehnel, Tim Hilden, Gabe Hoffmann, Burkhard Huhnke, Doug Johnston, Stefan Klumpp, Dirk Langer, Anthony Levandowski, Jesse Levinson, Julien Marcil, David Orenstein, Johannes Paefgen, Isaac Penny, Anna Petrovskaya, Mike Pflueger, Ganymed Stanek, David Stavens, Antone Vogt, and Sebastian Thrun, *Junior: The stanford entry in the urban challenge*, Journal of Field Robotics **25** (2008), no. 9, 569–597. (Cited on page 11.)

[MG09] Michael Müter and André Groll, *Attack detection for in-vehicle networks*, VDI Conference on Automotive Security, 2009. (Cited on pages 95, 115, and 159.)

[Mit03] C.J. Mitchell, *Truncation attacks on macs*, Electronics Letters **39** (2003), no. 20, 1439 – 1440. (Cited on page 69.)

[MTV09] Gianpaolo Macario, Marco Torchiano, and Massimo Violante, *An in-vehicle infotainment software architecture based on google android*, 2009 IEEE International Symposium on Industrial Embedded Systems (2009), 257–260. (Cited on pages 4 and 122.)

[MWW+11] Stuart McClure, Andre Weimerskirch, Marko Wolf, Christof Paar, Winfried Stephan, and Stefan Goss, *Caution: Malware Ahead. An analysis of emerging risks in automotive system security*, Tech. report, McAfee, Santa Clara, 2011. (Cited on pages 15 and 133.)

[Nec97] George C. Necula, *Proof-carrying code*, POPL '97: Proceedings of the 24th ACM SIGPLAN-SIGACT symposium on Principles of programming languages (New York, NY, USA), ACM, 1997, pp. 106–119. (Cited on page 23.)

[Nec98] ———, *Compiling with Proofs*, Ph.D. thesis, Carnegie Mellon University, 1998. (Cited on page 23.)

[Nis12] Nissan, *Nissan pioneers first-ever independent control steering technology*, Press Release, Oct. 17 2012. (Cited on pages 11 and 129.)

[NL98] George C. Necula and Peter Lee, *The design and implementation of a certifying compiler*, SIGPLAN Not. **33** (1998), no. 5, 333–344. (Cited on page 23.)

[NLJ08] Dennis K. Nilsson, Ulf E. Larson, and Erland Jonsson, *Efficient in-vehicle delayed data authentication based on compound message authentication codes*, Vehicular Technology Conference (VTC Fall '08), IEEE, 2008, pp. 1–5. (Cited on pages 15, 74, and 133.)

[NLPJ09] Dennis K. Nilsson, Ulf E. Larson, Francesco Picasso, and Erland Jonsson, *A first simulation of attacks in the automotive network communications protocol flexray*, Proceedings of the International Workshop on Computational Intelligence in Security for Information Systems CISIS'08 (Emilio Corchado, Rodolfo Zunino, Paolo Gastaldo, and Álvaro Herrero, eds.), Advances in Soft Computing, vol. 53, Springer Berlin Heidelberg, 2009, pp. 84–91. (Cited on pages 15 and 133.)

[NS05] James Newsome and Dawn Xiaodong Song, *Dynamic taint analysis for automatic detection, analysis, and signature generation of exploits on commodity software*, NDSS, The Internet Society, 2005. (Cited on page 94.)

[OBL+10] Kaan Onarlioglu, Leyla Bilge, Andrea Lanzi, Davide Balzarotti, and Engin Kirda, *G-free: Defeating return-oriented programming through gadget-less binaries*, 26th ACSAC, December 2010. (Cited on page 93.)

[Orm03] Hilarie Orman, *The Morris worm: A fifteen-year perspective*, IEEE Security & Privacy Magazine **1** (2003), no. 5, 35–43. (Cited on pages 36 and 138.)

[OYN+08] Hisashi Oguma, Akira Yoshioka, Makoto Nishikawa, Rie Shigetomi, Akira Otsuka, and Hideki Imai, *New Attestation Based Security Architecture for In-Vehicle Communication*, Global Telecommunications Conference, 2008. IEEE GLOBECOM 2008. IEEE, IEEE, 2008, pp. 1–6. (Cited on pages 15, 73, and 133.)

[Pap09] Panos Papadimitratos, *Secure vehicular communication systems: Towards deployment*, CAST Workshop on Mobile Security for Intelligent Cars (2009). (Cited on pages 14 and 131.)

[PBH+08] P. Papadimitratos, L. Buttyan, T. Holczer, E. Schoch, J. Freudiger, M. Raya, Z. Ma, F. Kargl, A. Kung, and J.-P. Hubaux, *Secure vehicular communication systems: design and architecture*, IEEE Communications Magazine **46** (2008), no. 11, 100–109. (Cited on pages 16, 94, and 129.)

[Pes97] Alexander Peslyak (Solar Designer), *Non-executable stack patch*, Linux Kernel Patch, 1997. (Cited on page 93.)

[PGW+11] Hagen Platzdasch, Timo Gendrullis, Marko Wolf, Christian Fischer, Wolfgang Wiewesiek, Hervé Seudé, Yves Roudier, Hendrik Schweppe, Gabriel Pedroza, and Ludovic Apvrille, *D4.0.3 security architecture implementation – progress report*, Tech. report, EVITA Project, 2011. (Cited on pages 52 and 143.)

[PLSP07] Axel Poschmann, Gregor Leander, Kai Schramm, and Christof Paar, *New light-weight crypto algorithms for rfid*, Circuits and Systems, 2007. ISCAS 2007. IEEE International Symposium on, may 2007, pp. 1843 –1846. (Cited on page 57.)

[PRE14] PRESERVE, *The PRESERVE project: Preparing secure vehicle-to-x communication systems*, online at http://preserve-project.eu/, 2011-2014. (Cited on pages 17, 117, 130, and 161.)

[Ray09] Maxime Raya, *Data-Centric Trust in Ephemeral Networks*, Ph.D. thesis, École Polytechnique Fédérake de Lausanne (EPFL), 2009, pp. 1–111. (Cited on pages 6, 26, and 125.)

[RH05] Maxim Raya and Jean-Pierre Hubaux, *The security of vehicular ad hoc networks*, Proceedings of the 3rd ACM workshop on Security of ad hoc and sensor networks (New York, NY, USA), SASN '05, ACM, 2005, pp. 11–21. (Cited on pages 1 and 120.)

[Rie09] Konrad Rieck, *Machine Learning for Application-Layer Intrusion Detection*, Phd thesis, TU Berlin, 2009. (Cited on page 77.)

[RMM+10] Ishtiaq Rouf, Rob Miller, Hossen Mustafa, Travis Taylor, Sangho Oh, Wenyuan Xu, Marco Gruteser, Wade Trappe, and Ivan Seskar, *Security and privacy vulnerabilities of in-car wireless networks: a tire pressure monitoring system case study*, Proceedings of the 19th USENIX conference on Security, 2010. (Cited on pages 13, 75, 83, 94, 153, and 155.)

[Rob91] Robert Bosch GmbH, *CAN specification v2.0*, September 1991. (Cited on pages 67 and 69.)

[RPB+09] Indrajit Roy, Donald E. Porter, Michael D. Bond, Kathryn S. McKinley, and Emmett Witchel, *Laminar: practical fine-grained decentralized information flow control*, Proceedings of the 2009 ACM SIGPLAN conference on Programming language design and implementation (New York, NY, USA), PLDI '09, ACM, 2009, pp. 63–74. (Cited on pages 28 and 137.)

[RRKH04] Srivaths Ravi, Anand Raghunathan, Paul Kocher, and Sunil Hattangady, *Security in embedded systems: Design challenges*, ACM Trans. Embed. Comput. Syst. **3** (2004), no. 3, 461–491. (Cited on pages 112, 113, and 157.)

[RWW+09] A. Ruddle, D. Ward, B. Weyl, S. Idrees, Y. Roudier, M. Friedewald, T. Leimbach, A. Fuchs, S. Gürgens, O. Henniger, R. Rieke, M. Ritscher, H. Broberg, L. Apvrille, R. Pacalet, and G. Pedroza, *Security requirements for automotive on-board networks based on dark-side scenarios*, Tech. Report Deliverable D2.3, EVITA Project, 2009. (Cited on pages 47, 48, 71, and 143.)

[scu01] scut, *Exploiting format string vulnerabilities*, Tech. report, TESO Security Group, 2001. (Cited on page 86.)

[SeV08] SeVeCom, *The SeVeCom project*, 2006—2008. (Cited on pages 14 and 131.)

[SIR+10] H. Schweppe, S. Idrees, Y. Roudier, B. Weyl, R. El Khayari, O. Henniger, D. Scheuermann, G. Pedroza, H. Seudié, and H. Platzdasch, *Secure on-board protocols specification*, Tech. Report Deliverable D3.3, EVITA Project, 2010. (Cited on pages 49, 58, 150, and 169.)

[SKL+06] Elmar Schoch, Frank Kargl, Tim Leinmüller, Stefan Schlott, and Panos Papadimitratos, *Impact of pseudonym changes on geographic routing in vanets*, Proceedings of the Third European conference on Security and Privacy in Ad-Hoc and Sensor Networks (Berlin, Heidelberg), ESAS'06, Springer-Verlag, 2006, pp. 43–57. (Cited on pages 16 and 129.)

[Sli12] Eric Slivka, *Apple pulls russian sms spam app from app store*, MacRumors news, online at http://www.macrumors.com/2012/07/05/apple-pulls-russian-sms-spam-app-from-app-store/, July 5 2012. (Cited on pages 116 and 160.)

[Slo12] Mary Slosson, *Google gets first self-driven car license in Nevada*, Reuters News, online at http://www.reuters.com/article/2012/05/08/uk-usa-nevada-google-idUSLNE84701320120508, May 8 2012. (Cited on page 11.)

[SPPG04] Hovav Shacham, Matthew Page, Ben Pfaff, and EJ Goh, *On the effectiveness of address-space randomization*, CCS, 2004. (Cited on pages 86 and 93.)

[SR12] Hendrik Schweppe and Yves Roudier, *Security and privacy for in-vehicle networks*, The First International Workshop on Vehicular Communications, Sensing, and Computing (VCSC 2012), June 2012. (Cited on page 95.)

[SRRS01] R. Sekar, C. R. Ramakrishnan, I. V. Ramakrishnan, and S. A. Smolka, *Model-carrying code (mcc): a new paradigm for mobile-code security*, NSPW '01: Proceedings of the 2001 workshop on New security paradigms (New York, NY, USA), ACM, 2001, pp. 23–30. (Cited on page 23.)

[SRW+11] Hendrik Schweppe, Yves Roudier, Benjamin Weyl, Ludovic Apvrille, and Dirk Scheuermann, *Car2X communication: securing the last meter - A cost-effective approach for ensuring trust in Car2X applications using in-vehicle symmetric cryptography*, WIVEC 2011, 4th IEEE International Symposium on Wireless Vehicular Communications, September 2011. (Cited on page 74.)

[SS11] Klaus Scheibert and Björn Steurich, *Sichere mikroprozessorarchitekturen (infineon technologies ag, in german)*, VDI 27. Gemeinschaftstagung on Automotive Security (2011), 17–34. (Cited on pages 117 and 161.)

[SVB+03] R. Sekar, V.N. Venkatakrishnan, Samik Basu, Sandeep Bhatkar, and Daniel C. DuVarney, *Model-carrying code: a practical approach for safe execution of untrusted applications*, SOSP '03: Proceedings of the nineteenth ACM symposium on Operating systems principles (New York, NY, USA), ACM, 2003, pp. 15–28. (Cited on pages 23 and 24.)

[Sys12] SysSec NoE, *D6.2: Intermediate report on the security of the connected car*, Tech. report, September 2012. (Cited on pages 14, 16, 131, and 133.)

[TIS] TISA, Traveller Information Services Association, *The traffic message channel (TMC). online at http://www.tisa.org/technologies/tmc/*. (Cited on page 12.)

[TMD+06] Sebastian Thrun, Mike Montemerlo, Hendrik Dahlkamp, David Stavens, Andrei Aron, James Diebel, Philip Fong, John Gale, Morgan Halpenny, Kenny Lau, Celia Oakley, Mark Palatucci, Vaughan Pratt, Pascal Stang, Sven Strohb, Cedric Dupont, Lars erik Jendrossek, Christian Koelen, Charles Markey, Carlo Rummel, Joe Van Niekerk, Eric Jensen, Gary Bradski, Bob Davies, Scott Ettinger, Adrian Kaehler, Ara Nefian, and Pamela Mahoney, *Stanley: The robot that won the darpa grand challenge: Research articles*, J. Robot. Syst. **23** (2006), no. 9, 661–692. (Cited on page 11.)

[Uni05] United Nations ECE, *Uniform provisions concerning the approval of vehicles with regard to steering equipment*, Agreement Addendum 78: Regulation No. 79 Revision 2, April 2005. (Cited on pages 11 and 129.)

[Uni10] _____, *Uniform provisions concerning the approval of vehicles with regard to the installation of lighting and light-signalling devices*, Agreement Addendum 47: Regulation No. 48 Revision 6, June 2010. (Cited on pages 40 and 145.)

[Uni11] _____, *Uniform provisions concerning the approval of vehicles of categories m, n and o with regard to braking*, Agreement Addendum 12: Regulation No. 13 Revision 7, August 2011. (Cited on pages 40 and 145.)

[US 09] US Department of Transportation, *Family of standards for wireless access in vehicular environments (WAVE)*, online: http://www.standards.its.dot.gov/fact%5Fsheet.asp?f=80, September 2009. (Cited on pages 16 and 132.)

[Wey08] Benjamin Weyl, *On Interdomain Security: Trust Establishment in Loosely Coupled Federated Environments*, Ph.D. thesis, TU Darmstadt, 14 December 2008, p. 260. (Cited on page 26.)

[WFLW08] Peng Wang, Dengguo Feng, Changlu Lin, and Wenling Wu, *Security of truncated macs*, Inscrypt (Moti Yung, Peng Liu, and Dongdai Lin, eds.), Lecture Notes in Computer Science, vol. 5487, Springer, 2008, pp. 96–114. (Cited on page 71.)

[WG11] Marko Wolf and Timo Gendrullis, *Design, implementation, and evaluation of a vehicular hardware security module*, ICISC (Howon Kim, ed.), Lecture Notes in Computer Science, vol. 7259, Springer, 2011, pp. 302–318. (Cited on pages 53, 54, 57, 117, 161, and 163.)

[Wil10] Martyn Williams, *Toyota to recall Prius hybrids over ABS software*, Computerworld (2010). (Cited on pages 11 and 128.)

[WLAG93] Robert Wahbe, Steven Lucco, Thomas E. Anderson, and Susan L. Graham, *Efficient software-based fault isolation*, Proceedings of the fourteenth ACM symposium on Operating systems principles (New York, NY, USA), SOSP '93, ACM, 1993, pp. 203–216. (Cited on page 22.)

[WWZ+10] B. Weyl, M. Wolf, F. Zweers, T. Gendrullis, M. S. Idrees, Y. Roudier, H. Schweppe, H. Platzdasch, R. El Khayari, O. Henniger, D. Scheuermann, A. Fuchs, L. Apvrille, G. Pedroza, H. Seudié, J. Shokrollahi, and A. Keil, *Secure on-board architecture specification*, Tech. Report Deliverable D3.2, EVITA Project, 2010. (Cited on pages 48, 49, 57, 71, 115, 143, and 159.)

[XBS06] Wei Xu, Sandeep Bhatkar, and R. Sekar, *Taint-enhanced policy enforcement: a practical approach to defeat a wide range of attacks*, Proceedings of the 15th conference on USENIX Security Symposium - Volume 15 (Berkeley, CA, USA), USENIX-SS'06, USENIX Association, 2006. (Cited on pages 28, 114, 137, and 158.)

[XH03] S Xia and J Hook, *Experience with Abstraction-carrying Code*, Electronic Notes in Theoretical Computer Science (2003). (Cited on page 24.)

[YSE+07] Heng Yin, Dawn Song, Manuel Egele, Christopher Kruegel, and Engin Kirda, *Panorama: capturing system-wide information flow for malware detection and analysis*, Proceedings of the 14th ACM conference on Computer and communications security, ACM, 2007, pp. 116–127. (Cited on page 94.)

[ZJS+09] Yu Zhu, Jaeyeon Jung, Dawn Song, Tadayoshi Kohno, and David Wetherall, *Privacy scope: A precise information flow tracking system for finding application leaks*, Tech. Report UCB/EECS-2009-145, EECS Department, University of California, Berkeley, Oct 2009. (Cited on pages 27 and 136.)

[ZJSK11] DY Zhu, J Jung, Dawn Song, and T Kohno, *TaintEraser: protecting sensitive data leaks using application-level taint tracking*, ACM SIGOPS Operating Systems Review (2011). (Cited on pages 84, 94, and 156.)

[ZPK11] Angeliki Zavou, Georgios Portokalidis, and Angelos D. Keromytis, *Taint-exchange: a generic system for cross-process and cross-host taint tracking*, Proceedings of the 6th International conference on Advances in information and computer security (Berlin, Heidelberg), IWSEC'11, Springer-Verlag, 2011, pp. 113–128. (Cited on pages 28, 94, 116, 138, and 160.)

[ZS08] Werner Zimmermann and Ralf Schmidgall, *Bussysteme in der Fahrzeugtechnik (in German)*, 3 ed., Vieweg-Teubner, 2008. (Cited on pages 15 and 132.)