

TECHNISCHE UNIVERSITÄT MÜNCHEN

Lehrstuhl für Sicherheit in der Informatik

**Middleware-based Security
for Future In-Car Networks**

Alexandre Bouard

Vollständiger Abdruck der von der Fakultät für Informatik der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften (Dr. rer. nat.)

genehmigten Dissertation.

Vorsitzender: Univ.-Prof. Dr. Uwe Baumgarten

Prüfer der Dissertation:

1. Univ.-Prof. Dr. Claudia Eckert
2. Prof. Refik Molva, Ph.D.
(EURECOM, Sophia Antipolis, France)

Die Dissertation wurde am 17.03.2014 bei der Technischen Universität München eingereicht und durch die Fakultät für Informatik am 21.07.2014 angenommen.

Foreword

This dissertation has been submitted in partial fulfillment of the requirements for the degree of Doktor der Naturwissenschaften at the department of Informatics, Technical University Munich (TUM). This study has been carried out in the period from October 2010 to August 2014.

Personal Acknowledgements

The last few years at the BMW Research and Technology Office of Munich have been very exciting, challenging and rewarding. Full of new perspectives, deadlines, accomplishments and memories, it has been an incredible experience to learn, develop and share. This thesis represents a substantial part of the research I conducted for my PhD. This would not have been possible without the people around me. I now would like to sincerely thank them.

First and foremost, I would like to thank my advisor, Prof. Claudia Eckert, not only for giving me the chance to do a Ph.D. at the TUM, but more importantly for supervising me during these years and providing me with enlightening discussions and helpful advice. My next sincere thanks go to my 'industrial' advisors: Dr. Daniel Herrscher for taking and trusting a young French student into his 'SEIS' team, Dr. Benjamin Weyl for his excellent guidance and fruitful comments, and Dr. Dennis Burgkhardt for his continuous support and useful tips. I would also like to express my gratitude to Prof. Ulrich Finger, director of EURECOM for encouraging me and giving me the contacts to start this journey.

Furthermore, my experience at BMW would have been as been the same without my awesome colleagues, co-authors and friends. Thank you for helping me navigate through the infinite processes and rules of BMW, for correcting my German, for having nice discussions and for all the fun.

At last, but not least, I want to express my deep gratitude to my family for their support and care throughout all these years. Special mention goes to my parents, Monique and Jean-Paul Bouard, for being there for me at all times and to my wife, Dr. Mariam

Kaynia, for being my true source of inspiration and motivation. Without you, I would not have achieved all these accomplishments and joyful memories.

Project Acknowledgements

The research presented here took place within the project SEIS - Security in Embedded IP-based Systems. The research project explored the usage of the Internet Protocol (IP) as a common and secure communication basis for electronic control units in vehicles. The project was partially funded by the German Federal Ministry of Education and Research (support codes 01BV0900 - 01BV0917). I would like to thank all SEIS partners directly or indirectly involved in this research.

München, August 29, 2014

Alexandre Bouard

Abstract

Each year, car manufacturers are competing to provide new and trendy automotive features for safety, driving assistance and infotainment. For this purpose, today's cars take advantage of powerful electronic platforms and embed more and more sophisticated connected services. More than just ensuring their role of a safe transportation mean, which remains nonetheless their primary function, cars have seen an extension of their paradigm towards the driving pleasure and the infotainment domain. Thus, they process large amounts of sensitive data, e.g., personal information, industrial secrets; they are increasingly tethered to the external environment via smartphones, Internet or other road-side units; and like the consumer electronics world, they will very soon host downloadable and on-the-fly installable Third-Party Applications (TPAs). However, the car pervasive computerization exposes them to unintentional programming bugs and to common security attacks targeting not only the data they contain but also their own integrity. Today, traditional automotive technologies cannot protect against any of these threats. Without any countermeasures, these security vulnerabilities could lead to unfortunate consequences: lawsuits, damages to the enterprise reputation, loss of billions of dollars, driving discomfort or even worse, endangering the life of the car passengers.

The transition towards Ethernet/Internet Protocol (IP)-based on-board communications and mature security protocols could be a first, but not sufficient, step to respond to these security and privacy issues. This thesis is in line with this evolution and focuses on the design and implementation of an automotive IP-based security middleware leveraging local and distributed information flow techniques in order to protect the car against on-board and external threats.

Unlike previous automotive approaches, security is defined and enforced at the middleware level. This approach allows to abstract the security interfaces and simplify its maintenance and verification. A suitable modularization eases the fulfillment of all security and functional requirements. A security architecture for middleware was developed within this thesis leveraging hardware security platforms. The middleware provides mechanisms for on-board and external secure communication channels as well as dedicated security decision- and enforcement-points.

In addition to just providing strong security between two on-board electronic platforms, an authorization model based on decentralized information flow control was further developed and integrated into the middleware layer. The model enforces label-based policies in order to follow the propagation of data of interest within the whole car and to safely and securely integrate untrustworthy use cases like smartphones or TPAs. An advanced approach based on dynamic data flow tracking was also investigated and coupled to the previous model. It provides mechanisms for deeper introspection and finer control of the TPA.

Then, a proof-of-concept implementation demonstrates the feasibility of the developed

security framework. It shows that the security risks can be mitigated via the middleware. Performance benchmarks demonstrate that such a middleware could provide a high throughput and relatively high bandwidth while ensuring the necessary security guarantees for authentication, encryption, integrity and authorization. However, this approach also showed its limits and requires the use of additional methods when a very high bandwidth is necessary.

As a conclusion, this thesis demonstrates that middleware-based security can be leveraged to achieve holistic security in cars. It provides the necessary general basis to build suitable security mechanisms adaptable to both threats and use cases. While these concepts were developed and assessed for an automotive context, they can also be extended to other demanding distributed systems, like trains, aircraft or smart buildings.

Zusammenfassung

Jedes Jahr bieten die Autohersteller neuere und modernere Fahrzeugfunktionen hinsichtlich Sicherheit, Fahrerassistenz und Infotainment an. Zu diesem Zweck nutzen heutige Fahrzeuge leistungsfähige elektronische Plattformen und betten immer mehr anspruchsvolle Funktionen ein. Fahrzeuge entwickelten sich dadurch von einfachen aber sicheren Verkehrsmitteln zu Statussymbolen für Fahrspaß und Infotainment. Dazu verarbeiten die Fahrzeuge große Mengen von sensiblen Daten, wie zum Beispiel persönliche Informationen oder Betriebsgeheimnisse. Dafür werden sie immer häufiger mit Smartphones, Internet oder anderen Infrastrukturen wie Ampeln, vernetzt. Schon bald werden darüber hinaus, wie in der "Consumer Electronic"-Welt, herunterladbare Anwendungen von Drittanbietern (TPAs) angeboten werden. Mit der allgegenwärtigen Integration von Computern ins Fahrzeug steigt jedoch die Gefahr von Programmierfehlern und Sicherheitsangriffen auf Fahrzeugdaten und deren Integrität. Allerdings bieten aktuelle Technologien in der Automobilindustrie keinen ausreichenden Schutz gegen diese Bedrohungen. Ohne die geeigneten Gegenmaßnahmen könnten Sicherheitslücken fatale Folgen haben. Neben Klagen, Schäden des Unternehmensrufs und finanziellem Verlust könnte sogar die Gesundheit der Insassen gefährdet werden.

Die Integration von Ethernet/Internet Protocol (IP)-basierter Bordnetzkommunikation und deren Sicherheitsprotokolle kann als erste, aber nicht ausreichende Maßnahme betrachtet werden, um auf diese Sicherheits- und Datenschutz-Probleme zu reagieren. Die vorliegende Arbeit fokussiert sich auf den Entwurf und die Umsetzung einer Fahrzeug-IP-basierten Sicherheits-Middleware, die lokale und verteilte Informationsflusstechniken nutzt, um das Auto gegen interne und externe Bedrohungen zu schützen.

Zu diesem Zweck werden Sicherheitsmechanismen auf Middleware-Ebene definiert und durchgesetzt. Dieser Ansatz erlaubt, Sicherheitsschnittstellen zu abstrahieren und deren Wartung und Verifikation zu vereinfachen. Eine geeignete Modularisierung erleichtert die Erfüllung von Sicherheits- und funktionalen Anforderungen. Darüber hinaus wurde im Rahmen dieser Arbeit eine Sicherheitsarchitektur für die Middleware entwickelt, welche Hardware-Security-Module beinhaltet. Die Middleware bietet Mechanismen für sichere Kommunikationskanäle, sowohl fahrzeugintern als auch -extern, sowie dedizierte *Security Decision* und *Enforcement Points* an.

Zusätzlich zu dieser Sicherheitsschicht auf der Kommunikationsebene wurde ein Autorisierungsmodell auf Basis einer *Decentralized Information Flow Control* entwickelt und in die Middleware-Schicht integriert. Das Modell setzt Label-basierte Regeln ein, um die Ausbreitung von relevanten Daten innerhalb des gesamten Fahrzeugs nachzuvollziehen. Dies ermöglicht die sichere Integration von nicht vertrauenswürdigen Anwendungsfällen, wie Smartphones oder TPAs. Ein weiterentwickelter Ansatz basierend auf *Dynamic Data Flow Tracking* wurde ebenfalls untersucht und in das bestehende Modell integriert. Die Anbindung ermöglicht Mechanismen zur präzisen Beobachtung und

Steuerung der TPAs.

Mittels einer *Proof-of-Concept* Implementierung wurde die Umsetzbarkeit des Sicherheitsframeworks gezeigt. Darüber hinaus zeigt die Implementierung, dass die Sicherheitsrisiken auf der Middleware-Ebene reduziert werden können. Mit Hilfe von Performanz-Benchmarks konnte gezeigt werden, dass eine solche Middleware-basierte Lösung, trotz hohem Sicherheitslevel einen hohen Durchsatz und eine relativ große Bandbreite ermöglicht. Bei extrem hohen Bandbreiten zeigt sich allerdings, dass zusätzliche Methoden erforderlich sind.

In Summe zeigt diese Arbeit, dass middlewarebasierte Sicherheitslösungen genutzt werden können, um ganzheitliche Sicherheit im Auto zu gewährleisten. Solche Lösungen bieten die notwendigen technischen Grundlagen, um an Bedrohungen und Anwendungsfälle angepasste Sicherheitsmaßnahmen umzusetzen. Obwohl diese Sicherheitskonzepte im automobilen Kontext entwickelt und bewertet wurden, können diese auch für andere anspruchsvolle verteilte Systeme, wie z.B. Züge, Flugzeuge oder intelligente Gebäude, adaptiert werden.

Résumé

Chaque année les constructeurs automobiles, rivalisant d'ingéniosité, proposent de nouvelles fonctionnalités innovantes assurant la sécurité, l'aide à la conduite et l'infodivertissement. A cette fin, les voitures profitent aujourd'hui de plates-formes électroniques performantes permettant l'intégration de services connectés de plus en plus sophistiqués. En plus d'assurer la fonction première de la voiture, à savoir être un moyen de transport sûr, le paradigme associé à celle-ci couvre désormais les aspects liés au plaisir de la conduite et au domaine de l'infodivertissement. Ainsi aujourd'hui, les voitures traitent de grandes quantités d'informations sensibles, par exemple, des données privées ou des secrets industriels; elles sont de plus en plus connectées au monde extérieur par le biais de smartphones, d'Internet ou d'autres infrastructures présentes en bord de route; et très bientôt, elles pourront héberger des applications tiers (TPA) téléchargeables et installables à la volée. Néanmoins, l'informatisation de celles-ci qui progresse irrémédiablement les expose à des défaillances liées à des bugs de programmation ou à des attaques informatiques qui non seulement ciblent les données que la voiture peut contenir, mais aussi son intégrité de fonctionnement. Aujourd'hui, les technologies traditionnellement utilisées dans le monde de l'automobile ne peuvent pas les protéger efficacement contre ces menaces. Sans protection, ces failles de sécurité peuvent avoir des conséquences fâcheuses : poursuites judiciaires, dégradation de la réputation de l'entreprise, perte financière, inconfort de conduite ou même pire, mise en danger de la vie des passagers.

L'introduction de communications de bord basées sur l'Ethernet/Internet Protocol (IP) et sur des protocoles de sécurité éprouvés peut être vue comme un premier pas, mais reste insuffisante pour répondre à toutes les questions relatives à la sécurité de la voiture et à la protection de la vie privée des utilisateurs. Cette thèse s'inscrit dans ce contexte de changement et porte essentiellement sur la conception et la mise en œuvre d'un middleware de sécurité pour les voitures utilisant l'Ethernet/IP. Ce middleware tire avantage de méthodes de contrôle de flux d'informations appliquées de manière locale et distribuée afin de protéger la voiture contre tout type de menaces internes et externes.

Ainsi dans ce contexte, la sécurité est définie et appliquée au niveau du middleware. Cette approche permet de définir par abstraction les interfaces de sécurité et de simplifier leur maintenance et leur vérification. Une définition appropriée des modules facilite la satisfaction de toutes les exigences requises en termes de fonctionnalité et de sécurité. Une architecture de sécurité du middleware s'appuyant sur des plates-formes de sécurité matériel (HSM) a été développée spécifiquement dans le cadre de cette thèse. Le middleware permet de développer des mécanismes pour l'établissement de canaux de communication sécurisés internes et externes, ainsi que pour des interfaces dédiées à la résolution et l'application des décisions de sécurité.

Il a été possible de développer et d'intégrer un modèle d'autorisation basé sur des règles de *Decentralized Information Flow Control* au sein de la couche middleware, tout

en fournissant une solution sécurisée pour les communications entre deux plates-formes. Le modèle applique des règles de labels qui permettent de suivre la propagation de données sensibles à bord de la voiture et d'intégrer en toute sécurité des cas d'utilisation risqués comme ceux de smartphones ou de TPAs. Une approche plus approfondie fondée sur les principes de *Dynamic Data Flow Tracking* a également été étudiée et associée au modèle précédent. Il permet de fournir des mécanismes de sécurité pour une introspection plus profonde et un contrôle plus affiné de la TPA.

Ensuite, la mise en œuvre du système de sécurité objet de cette thèse a montré la faisabilité de la réalisation de celui-ci et de son exploitation. Elle démontre ainsi que les risques de sécurité peuvent être atténués en regroupant les mécanismes de sécurité au niveau du middleware. Des tests de performance montrent par ailleurs qu'un tel middleware peut fournir un débit de requête élevé et une bande passante relativement large tout en assurant les garanties de sécurité nécessaires. Cependant, cette approche a aussi montré ses faiblesses et nécessite l'utilisation de méthodes additionnelles, lorsqu'une bande passante très élevée est requise.

En conclusion, cette thèse démontre que la sécurité au niveau du middleware peut être mise à profit pour atteindre un état de sécurité holistique au sein de la voiture. Elle définit des bases techniques générales et nécessaires pour mettre en place des mécanismes de sécurité appropriés qui peuvent s'adapter à la fois aux cas d'utilisation et aux menaces rencontrées. Bien que ces concepts aient été développés et évalués pour un contexte lié à l'automobile, ils peuvent également être étendus à d'autres systèmes distribués nécessitant des niveaux de sécurité similaires, comme par exemple les trains, les avions ou les bâtiments intelligents.

Contents

Abstract	vii
Zusammenfassung	ix
Résumé	xi
1 Introduction	1
1.1 Motivation	1
1.2 Goals & Approach	3
1.3 Contributions	4
1.4 Outline	6
2 Automotive On-board Architecture and Investigated Scenarios	9
2.1 About Today's Car	9
2.1.1 On-board communications	10
2.1.2 C2X communications	13
2.1.3 Security Research	15
2.2 About Tomorrow's Car	18
2.2.1 The Future On-board Network	19
2.2.2 The Future On-board Communication Protocols	21
2.2.3 Securing the Future On-board Communication Protocols	23
2.2.4 The Future Multi-platform Antenna	26
2.3 Security Threats and Risk Analysis	27
2.3.1 The Attackers	27
2.3.2 Their Motivations	28
2.3.3 The Threats That Can Be Leveraged	29
2.3.4 The Attacker Model	35
2.4 Automotive Functional Requirements	36
2.5 Summary	37

3	Automotive IP-based Security Architecture	39
3.1	Middleware Security	39
3.1.1	Automotive Middleware	40
3.1.2	Security Middleware Extension (SME)	42
3.1.3	Functional Use Case and SME Management	54
3.2	Security Communication Proxy	58
3.2.1	Towards Secure Automotive Proxy-Middleware	59
3.2.2	Information Flow Control, a First Approach	60
3.2.3	Extending the SME for a Security Communication Proxy	61
3.2.4	Security & Trust Level (STL) Taxonomy	63
3.3	Middleware and Security Discussion	66
3.3.1	About the SME Architecture	67
3.3.2	About the Security Proxy Architecture	69
3.3.3	About the STL approach	70
3.3.4	Security Gains	70
3.4	Summary	72
4	Information Flow Control in Cars	73
4.1	Decentralized Information Flow Control (DIFC)	74
4.1.1	DIFC Related Work	74
4.1.2	DIFC Model	77
4.1.3	DIFC-enabled Middleware	80
4.1.4	Discussion	84
4.1.5	Conclusion	86
4.2	Dynamic Data Flow Tracking (DDFT)	86
4.2.1	DDFT Related Work	87
4.2.2	Tracking and Controlling the Execution via DDFT	88
4.2.3	Middleware-based propagation of DDFT taints	90
4.2.4	Discussion	92
4.2.5	Conclusion	93
4.3	Combining DIFC/DDFT	94
4.3.1	DIFC/DDFT-enabled Middleware	94
4.3.2	Discussion	97
4.3.3	Conclusion	101
4.4	Summary	101
5	Prototypical Evaluation and Discussion	105
5.1	Evaluation Methodology	105
5.1.1	Functional Evaluation of a Secure Runtime	106
5.1.2	Testing Environment	108
5.1.3	Engineering-driven Middleware Development and Setup	108

5.2	Middleware	109
5.2.1	Etch Middleware	110
5.2.2	Performance Results & Interpretation	114
5.3	Security Communication Proxy	115
5.3.1	Etch Proxy	115
5.3.2	Performance Results & Interpretation	117
5.4	Monitoring & Controlling the TPA	118
5.4.1	Isolation and Virtualization	119
5.4.2	DDFT Engine	120
5.4.3	TPA monitoring evaluation	121
5.5	Discussion	127
5.6	Summary	130
6	Conclusion and Outlook	131
6.1	Summary and Conclusion	131
6.2	Outlook and Implications	134
	Acronyms	137
	Bibliography	143
	A Appendix	163
A.1	Numerical values of Figure 5.5	163
A.2	Numerical values of Figure 5.6	163
A.3	Numerical values of Figure 5.7	164
	Curriculum Vitae	165

List of Figures

2.1	Schematic of a modern on-board network.	10
2.2	Schematic of a future on-board network.	20
2.3	Automotive protocol suite.	22
2.4	Considered use cases.	31
2.5	A concrete big use case.	33
3.1	Overview of an automotive IP-based middleware and its integration within the ISO/OSI model.	42
3.2	Connections between functional middleware and SME.	44
3.3	Functional use case: Open connection & data sending (client side).	56
3.4	Functional use case: Open connection & data sending (server side).	57
3.5	STL life cycle.	61
3.6	Binary decision tree for TL evaluation.	65
3.7	STL vector and main evaluation criteria.	66
4.1	Label-based lattice.	78
4.2	Example of label usage.	80
4.3	Overview of the DIFC-enabled middleware architecture.	81
4.4	Example of on-board label distribution.	82
4.5	DIFC-enabled automotive scenario.	84
4.6	Example of code with data dependencies and taint propagation.	89
4.7	Overview of the DDFT framework in the on-board network.	91
4.8	Architecture for DIFC/DDFT coupling.	95
5.1	Architecture of the <i>Etch</i> Java-binding.	111
5.2	Header serialization & in-band protocol of the <i>Etch</i> middleware.	113
5.3	Architecture of the <i>Etch</i> -enabled communication proxy.	116
5.4	Schematic view of a HU architecture leveraging virtualization and middleware for a secure TPA integration.	119
5.5	Throughput and bandwidth performance of the Client–Server scenario.	122

List of Figures

5.6	Throughput and bandwidth performance of the CE device–Proxy–HU–TPA scenario.	124
5.7	Throughput and bandwidth performance of the CE device–Proxy–TPA scenario.	126

List of Tables

2.1	Comparison between several automotive bus technologies.	11
2.2	Comparison of different protocols for securing the on-board communications.	24
2.3	The attackers and their motivations.	30
3.1	Partial API of the CSM.	46
3.2	Partial API of the KMM.	48
3.3	Partial API of the SCM.	49
3.4	Partial API of the AMM.	51
3.5	Partial API of the PMM.	52
3.6	Partial API of the IDM.	54
3.7	C2X Scenarios and assigned TL.	65
3.8	SME specifications.	68
4.1	Table comparing the three IFC approaches of Chapter 4.	99
4.2	Table comparing the three IFC approaches for attacks on the access control.	102
4.3	Table comparing the three IFC approaches for TPA-based attacks.	103
5.1	Throughput performance of the <i>Etch</i> middleware.	115
5.2	Throughput performance of the <i>Etch</i> proxy.	118
5.3	Normalized throughput performance of the Client–Server scenario.	123
5.4	Normalized throughput performance of the CE device–Proxy–HU–TPA scenario.	125
5.5	Normalized throughput performance of the CE device–Proxy–TPA scenario.	127

Listings

5.1	Definition of a <i>Etch</i> IDL file with specification of security metadata . . .	114
-----	--	-----

Chapter 1

Introduction

1.1 Motivation

During the last two decades, vehicles have evolved into very complex systems embedding powerful electronic platforms for various purposes, e.g., safety, driving assistance, infotainment. While still fulfilling their primary goal as safe transportation means, cars are now offering a plethora of new connectivity interfaces and communicate with numerous external communication partners: Internet, Consumer Electronic (CE) devices, Road-Side Units (RSUs) and other cars [53]. Besides, like smartphones, cars will soon host Third-Party Applications (TPAs) [115]. Such connectivity capacities and new application features will obviously allow a better car customization and a stronger tethering between all on-board and external communication partners. But on the other hand, they may raise the threat level and increase the attack likeliness via these newly extended communication interfaces.

Recently, cars have been shown to be vulnerable to simple attacks involving packet sniffing/injection and more complex ones, like buffer overflows. Those were performed by attackers having physical access to the car and its on-board network [102], but later ones have shown the feasibility to compromise some cars through most of their external communication interfaces [35, 152]. Then, today's automotive applications are mostly developed for a specific platform and for a precise car model. Car manufacturers generally know their developers and can therefore contractually set certain responsibilities and testing processes. While not providing an entirely perfect security, such a strategy allows car makers to keep the application integration process under their control. Loadable and on-the-fly installable applications revolutionized the CE world but may shake up the static architecture of the car. While being mostly intended for the infotainment purpose, TPAs, CE-based applications and other online services will have full access to the Internet, to several on-board functions and may secretly compromise the car integrity, steal the car manufacturer intellectual property and leak the driver's private data [164].

At a functional level, limited communication technologies (e.g., Controller Area Network (CAN), Media Oriented Systems Transport (MOST)) and drastic requirements for low latency and high robustness left only very little space to security. Part of the solution seems to lie in the use of Ethernet and the Internet Protocol (IP) as a standard for on-board communication [63]. A larger bandwidth and mature security protocols will allow to secure the communications between two on-board platforms, but may remain insufficient for consistent access control mechanisms in order to achieve a holistic security solution. Not considering the whole information security problem, i.e., how information travels through the system, may lead to privacy breaches and, even worse, to safety malfunctionings, which could endanger the passengers' life. From this situation, several challenges can be formulated.

First, future automotive applications will become more and more complex. They will be as demanding in terms of performance and robustness as today, remain distributed between several on-board platforms and exchange very large objects like environment models [174]. Like today, they will simultaneously trigger safety and infotainment functionalities and handle large amounts of sensitive data for drivers and car manufacturers. Thus all on-board functions and data should be protected accordingly against any malicious mischief.

Then, cars are expected to increasingly use all their Car-to-X (C2X) communication interfaces. Until now, their communication capacities were quite limited, developed and maintained under the control of the car manufacturer. But future external communication partners will soon be able to communicate directly with the on-board network. Of course, adding encryption and authentication will easily prevent several attacks like eavesdropping of the on-board network or addition of a new on-board node. But this will not solve the whole access control problem. Smartphones, online services and TPAs are potentially malicious, but may still be authenticated and authorized to communicate with the car. Thus they should not get access to all on-board resources, e.g., functions, data, memory, computation power. In addition, the access control mechanisms should be distributed over all on-board platforms.

Finally, cars currently benefit from the lack of transparency of their technologies and their limited external interfaces. Most security investigations were recently performed by academia and published on locations with very research-oriented impact. Combining more efficient C2X interfaces with the introduction of well-known technologies, like Ethernet, IP and other commodity platforms, may change the situation and attract more attackers.

As a consequence and in order to keep on producing safe and trustworthy vehicles, car manufacturers need to secure the on-board architecture accordingly and to design on-board software bases that can efficiently manage all on-board secure communications, leverage secure hardware features and enforce consistent access control mechanisms.

1.2 Goals & Approach

Automotive security cannot be handled like traditional IT security. Cars are sophisticated systems embedding millions of lines of code [34] and subject to very high functional requirements for high performance and robustness. From this situation, several goals for this thesis can be identified and are listed below:

Security goal: The security architecture should leverage the transition of the car towards an Ethernet/IP on-board network. More than just providing communication security, the security architecture should define a security management model for on-board and external access of on-board functions and data. Finally the resulting framework should ensure a safe and secure integration of untrustworthy external (e. g., CE device, online service) and on-board (e. g., TPA) communication partners.

Engineering goal: The development and management of security mechanisms should follow an engineering-driven model. The addition of security should not significantly increase the software complexity. The verification and maintenance of the secure on-board architecture should be efficient and potentially managed by a limited group of persons.

Functionality goal: Considering that the lives of the car passengers are at stakes, the security-enabled car should provide equivalent functional and safety performance as today. The addition of security within future Ethernet/IP-based on-board network should not add more latency or result in a higher risk of error.

Regarding the thesis approach, the first task is to analyze the current weaknesses and drawbacks of the current on-board network architecture. A second analysis of the Ethernet/IP architecture then highlights which communication protocols may be used and which kind of immediate security measures may be applied, i. e., existing protocols, standards. After this analysis phase, the definition of a secure software architecture may start. Considering the number of cases to handle, i. e., protecting the on-board functions, data and managing the communication security, the security should be application-independent. The middleware layer is like a Swiss army knife when it comes to manage communications from a functional point of view and is already extensively investigated and developed for the automotive purpose. The flexibility of this piece of software should be and will be leveraged for security between on-board platforms and for integrating external and on-board untrustworthy use cases. However, a consistent distributed authorization model for on-board function access and data management should also be specified. For this purpose, a formal access control model should abstract all information exchanges as basic flows and provide an efficient enforcement of security rules ensuring the car integrity and the data confidentiality. Regarding the other on-board untrusted components (i. e., TPAs), more intrusive methods enforced at a local level providing full control should also be investigated and integrated. Finally, a proof of concept should attest of the feasibility of all these concepts performing together and conclude, whether this thesis fulfills its goals.

1.3 Contributions

This thesis investigates the earlier mentioned security challenges and also demonstrates how to enforce security at the middleware level in an automotive context. The proposed middleware architecture provides an efficient on-board security management as well as a secure integration of very untrusted scenarios on which the car manufacturer has no control. More than just focusing on security, this architecture takes into account the automotive specifications of the whole car life cycle including the runtime but also the development phase. Despite the increasing car sophistication, the middleware-based approach abstracts all communications and security mechanisms and enables an easy maintenance and verification of the software components.

In short, the major contributions of this thesis are fourfold:

1. *Security Middleware Extension (SME)*. The SME architecture ensures security flexibility and does not depend on a specific middleware, i. e., it can be integrated in a large number of middleware architecture. It includes all the necessary mechanisms to manage the establishment of on-board secure communication channels and access control decision- and enforcement points for on-board functions and data management. Its modularization allows to customize the middleware layer depending on the asset to protect, on the platform capacities and, if present, to leverage secure hardware solutions.
2. *Security Communication Proxy*. All C2X communications are decoupled between external and on-board networks at the communication proxy level installed on C2X antennae, on the edge of the on-board network. As first and last communication barrier, the proxy enforces filtering of inbound and outbound messages and communicates with on-board components via a middleware-based in-band protocol. More than just a filter, the proxy assesses on-the-fly the security and trust level of the communication and of its participants based on an adapted taxonomy.
3. *Automotive Decentralized Information Flow Control (DIFC)*. The extended DIFC model enforces a car-wide consistent access control for a secure on-board data management and function access. Instead of defining numerous policies, all network exchanges are abstracted as flows of information. Each asset is assigned a label, on which a limited number of policies are enforced within the middleware. Labels and policies therefore allow to monitor, process or block flows of information directly from the middleware level. Additionally, such abstraction enables an easy and secure integration of untrustworthy on-board and external components.
4. *Automotive Dynamic Data Flow Tracking (DDFT)*. Extended DDFT mechanisms can also ease the integration of TPAs. The DDFT customization possibilities and its fine-grained enforcement ensure a total control over untrusted applications. Besides a middleware-based coupling via the proxy and the DIFC model allows the DDFT-monitored TPA to stay fully functional while fully controlling it, i. e., controlling what it receives, gets access to and sends on the network.

In the course of the research done in this thesis, the following papers were published:

1. Benjamin Weyl, Maximilian Graf and Alexandre Bouard: **Smart Apps in einem vernetzten (auto)-mobilen Umfeld: IT-Security und Privacy**. *Chapter in the book Smart Mobile Apps*, 2012 [180]
2. Alexandre Bouard, Johannes Schanda, Daniel Herrscher and Claudia Eckert: **Automotive Proxy-based Security Architecture for CE Device Integration**. In Proceedings of the 5th International Conference on Mobile Wireless Middleware, Operating Systems and Applications (MOBILEWARE '12), Berlin, Germany, November 13–14, 2012 [27]
3. Alexandre Bouard, Benjamin Glas, Anke Jentzsch, Alexander Kiening, Thomas Kittel, Franz Stadler and Benjamin Weyl: **Driving Automotive Middleware Towards a Secure IP-based Future**. In Proceedings of the 10th International Conference on Embedded Security in Cars (ESCAR '12), Berlin, Germany, November 28–29, 2012 [24]
4. Florian Sagstetter, Martin Lukasiewicz, Sebastian Steinhorst, Marko Wolf, Alexandre Bouard, Willian R. Harris, Someh Jha, Thomas Peyrin, Axel Poschmann and Samarjit Charaborty: **Security Challenges in Automotive Hardware/Software Architecture Design**, In Proceedings of the 16th International Conference on Design, Automation & Test in Europe (DATE '13), Grenoble, France, March 18–19, 2013 [154]
5. Alexandre Bouard, Maximilian Graf and Dennis Burgkhardt.: **Middleware-based Security & Privacy for In-car Integration of Third-party Applications**. In Proceedings of the 7th IFIP WG 11.11 International Conference on Trust Management (IFIP TM '13), Málaga, Spain, June 3–7, 2013 [26]
6. Alexandre Bouard, Hendrik Schweppe, Benjamin Weyl and Claudia Eckert: **Leveraging In-Car Security by Combining Information Flow Monitoring Techniques**. In Proceedings of the 2nd International Conference on Advances in Vehicular Systems, Technologies and Applications (VEHICULAR '13), Nice, France, July 21–25, 2013 [28]
7. Alexandre Bouard, Benjamin Weyl and Claudia Eckert: **Practical Information-Flow Aware Middleware for In-Car Communication**. In Proceedings of the 1st International Academic Workshop on Security, Privacy and dependability for CyberVehicles (CyCar '13 Co-located with CSS '13), Berlin, Germany, November 4, 2013 [29]
8. Alexandre Bouard, Dennis Burgkhardt and Claudia Eckert: **Middleware-based Security for Hyperconnected Applications in Future In-Car Networks**, In

EAI Endorsed Transactions on Mobile Communications and Applications Journal,
Vol. 13, Num. 3, December, 2013 [23]

1.4 Outline

Based on the approach in Section 1.2, this thesis consists of six chapters: an introduction (Chapter 1), a background overview (Chapter 2), three main concepts and evaluation chapters (Chapters 3 to 5) and a summary (Chapter 6). Considering the disparity of the considered topics, all chapters also include their own specialized related work.

Chapter 1. Introduction. This first chapter provides a motivation overview and gives a first insight about today's cars and their future evolution. In this context, problem statement as well as goals and approach are clearly stated. The main contributions of this work and a brief outline are also then formulated.

Chapter 2. Automotive On-board Architecture and Investigated Scenarios. This chapter provides a detailed description of the car Electrical/Electronic (E/E) architecture and some related work. Firstly, today's architecture and shortcomings are presented and discussed. Then, tomorrow's architecture and security benefits are looked into and set the context of this work. After what, an analysis of forthcoming threats is provided and thoroughly describes the automotive surface of attack, the different classes of attackers and their motivations. This section also leads to a formal definition of the attacker model followed by this thesis. Finally some additional requirements related to the automotive world are formulated as well.

Chapter 3. Automotive IP-based Security Architecture. This chapter presents a secure middleware architecture for future on-board communication infrastructures. First the modularization of the security middleware extension is described. Part of the Application Programming Interface (API) of each module is provided and presented in situ within a functional use case illustrating the interactions between security modules and functional middleware. Then an architecture for a secure C2X communication proxy and a first approach for Information Flow Control (IFC) are presented. The IFC is supported by the definition of a practical taxonomy allowing to determine the security and trust level of C2X communications. This chapter ends with an intermediary discussion debating the benefits of securing a system at the middleware level in this manner and its limits.

Chapter 4. Information Flow control in Cars. This chapter presents an car-wide authorization model enforced at the middleware level. A formal access control model based on Decentralized Information Flow Control (DIFC) is formalized and extended for a secure management of on-board communications and a secure integration of external C2X communicating partners and TPAs. Then an advanced approach for TPA integration based on Dynamic Data Flow Tracking (DDFT) is also investigated and is coupled to IFC model presented in Chapter 3. After what, a third approach is described and proposes to leverage both DIFC- and DDFT-based models thanks to

adapted security interfaces through which information can be exchanged. This chapter finally ends with a discussion comparing the resulting IFC models and provides some first recommendations.

Chapter 5. Prototypical Evaluation and Discussion. This chapter addresses the implementation implications of the concepts presented in the chapters 3 and 4. First the evaluation methodology is introduced and discusses the different factors to assess and the testing environments. It also provides more information about the automotive software development and the car life cycle. The proof-of-concept developed for this thesis is then presented; this includes the descriptions of the used technologies, the developed modules and their integration with each other. At the same time, the performance results of the different implemented components are listed and discussed. Finally this chapter ends with a final discussion putting in parallel security and functional evaluations. Some additional recommendations and system limitations are also debated here.

Chapter 6. Conclusion and Summary. This last chapter concludes the thesis. The major contributions are summarized here and discussed to highlight how they fulfill the goals and requirements defined in Chapters 1 and 2. An outlook is also provided and presents some directions that should also be investigated.

Chapter 2

Automotive On-board Architecture and Investigated Scenarios

Cars became very complex goods, which experience today multiple increasing security issues. This background chapter aims at providing a description of the Electrical/Electronic (E/E) architectures of both current and future vehicles and highlighting the major security threats to consider. It also proposes to discuss in details the security aspects of the four main scenarios investigated by this thesis: the on-board communications, the integration of online services, of CE devices and of on-board Third-Party Applications (TPAs). This chapter includes a significant part of the related work of this document. But considering the disparity of the considered topics, all following chapters also include their own specialized related work.

The on-board architecture of modern cars is discussed in Section 2.1. Afterwards, a potential architecture for future on-board Ethernet/IP-based network is debated in Section 2.2. Then the attacker model and threats considered by this thesis are presented in Section 2.3. Finally, Section 2.4 describes in more details the different requirements in term of functionality and security that this work should consider.

2.1 About Today's Car

Until a few years ago, vehicular communications only meant in-vehicle or on-board communications, i. e., between internal car components. But with the multiplication of interfaces communicating with the outside, C2X communications became as essential and critical as the on-board ones. From proprietary interfaces for diagnosis, these C2X interfaces are now leveraging all the potential of the GSM, the 3G, the wave band around 5,9 GHz as proposed by the IEEE 802.11p standard [1] and very soon LTE.

This section is structured in three points: (1) the in-vehicle communications, (2) the C2X communications and (3) some related work about automotive-specific security and research projects. While the two first parts describe the situation and related security

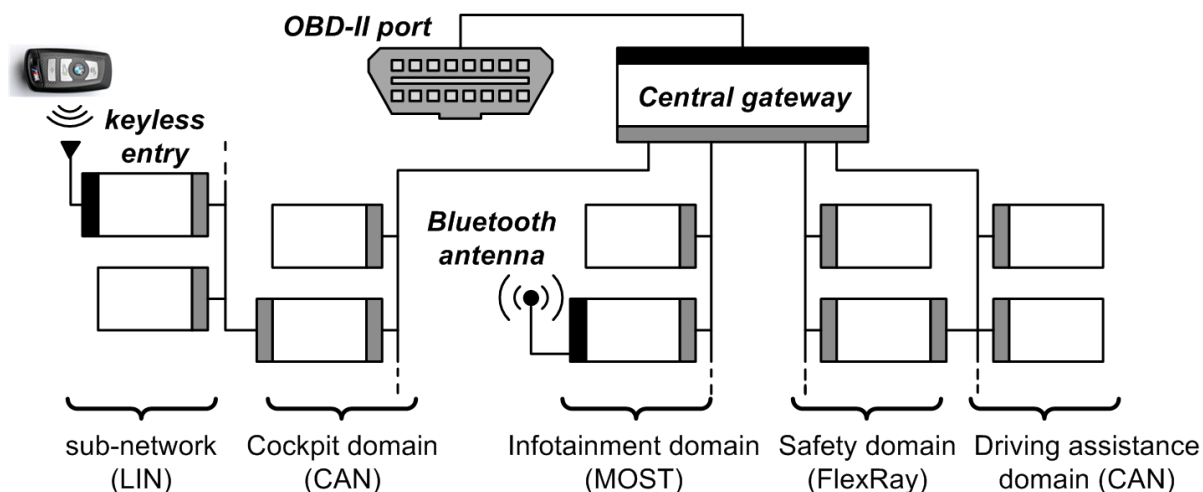





Figure 2.1: Schematic of a modern on-board network. The ECUs () are connected by buses. Gateway ECU may connect several buses with each other () or may be connected to C2X interfaces such as the OBD-II port or some wireless antennae (). The network topology of these bus technologies has not been respected.

issues, the third part lists a set of actions that academia and industry investigated and developed to enhance the car security.

2.1.1 On-board communications

This subsection is divided in two: (1) a precise description of the on-board communication infrastructure of modern cars and (2) the resulting security issues.

2.1.1.1 On-board Network

Today, the on-board network of premium vehicles has grown in a heterogeneous communication infrastructure including up to 80 interconnected Electronic Control Units (ECUs). As shown in Figure 2.1, a large variety of networking technologies allows to organize the ECUs around specific domains and results in a complex and cost-intensive architecture: several CAN [33] or Local Interconnect network (LIN) [110] subnets, a FlexRay subnet [59], a MOST subnet [125] and sometimes an Ethernet one [63]. These bus technologies come with their own custom protocols, which are not directly interoperable. As a consequence, dedicated application layer gateways are needed to interconnect them. But in time of increasing need for inter-domain communications and more bandwidth, such gateways became an obstacle for innovation.

Table 2.1 provides more information about all mentioned bus technologies. CAN is predominant in the on-board network. It establishes communications via broadcasted

Table 2.1: Comparison between several automotive bus technologies. “Possible harm” describes the consequences of an issue on such bus (error or attack).

	Low-CAN	High-CAN	FlexRay	LIN	MOST
Domain	body	body, power train, chassis	power train, chassis, safety	power train, chassis	telematics, multimedia
Standard	ISO 11519-2	ISO 1198	FlexRay consortium	LIN consortium	MOST consortium
Max. Bandwidth	125 kbit/s	1 Mbit/s	20 Mbit/s	19,2 kbit/s	25-150 Mbit/s
Topology	bus	bus	star	bus	ring
Max. number of nodes	24	10	22 per star	16	64
Applications	lights, wipers	engine, transmission	airbags	windows, doors	CD/DVD player
Control mechanism	event-driven	event-driven	time/event-driven	time-driven	time/event-driven
Possible harm		Risk of accident		Loss of functionality	Data theft
		Loss of assistance or control			Loss of comfort

and 8-bytes-long signals and uses a priority-based arbitration. This communication paradigm makes it suitable for messages sent to multiple recipients, even if it is rarely the case in practice. Then, MOST connects the infotainment domain and is used for complex tasks. It allows to exchange larger frames and exists in 3 versions, i. e., MOST25, MOST50, MOST150 providing a bandwidth of, respectively 25, 50 and 150 Mbit/s. Although MOST150 is a mature technology, its integration is still in progress for most car manufacturers. For applications with high bandwidth and hard real-time requirements, FlexRay proposes a scheduling featuring both static and dynamic communication slots assigned for all applications. The static slots provide very deterministic communications and allow to meet the real-time requirements. Finally, LIN is mostly used to connect small ECUs and sensors with limited criticality, even if a functional issue (or an attack) on the door locking could have dramatic consequences, such as the theft of the vehicle or to lock in the car occupants.

On-board automotive applications are divided into elementary blocks over diverse ECUs. They can usually rely on engineering-driven middleware in order to exchange messages. AUTOSAR [32] provides a common interface for signal-based CAN and FlexRay messages. On the other side, MOST proposes a very sophisticated middleware based on function blocks and which follows the principle of Remote Procedure Call (RPC).

The automotive industry also started to use Ethernet as communication media for high-end use cases. Ethernet in the APIX2 solution allows to establish transmission channels for audio and video between the Head Unit (HU), i. e., the central infotainment platform of the car, and the Rear Seat Entertainment (RSE) system [89]. Car manufacturers also started to use Ethernet for diagnosis purpose and to connect ECUs requiring large data transfer for initialization or update, e. g., the HU, the navigation ECU [14].

2.1.1.2 Security Issues

For a long time automotive security has been limited to anti-theft devices such as immobilizers and secure RFID transponders for car key. But recently, several research works highlighted numerous security issues due to a lack of protection on the communication buses and to poor ECU implementations. These issues and an increasing use of C2X communications reoriented the academic and industry research toward more holistic security solutions. Here follows a more detailed list of these issues. This non-exhaustive list focuses on the intrinsic problem of the on-board network. Most mentioned attacks [102, 77] have been performed through the OBD-II port or through open physical interfaces, i. e., without any intrusive methods. Attacks on external communication interfaces of the car are mentioned in the next subsection and some related work about automotive security is provided later in Section 2.1.3.

Insufficient bus security: None of the legacy automotive buses were designed with security in mind. They all lack the necessary protection mechanisms to provide authentication, integrity or confidentiality [75]. Performance requirements and the length of

a CAN packet, for example, do not allow to perform encryption or the addition of a Message Authentication Code (MAC) or of a digital signature. With CAN, all packets are received by every node, allowing all nodes to sniff and inject any type of packet and making a secure key distribution mechanism even harder to setup. Similarly the protocol used over CAN or FlexRay do not provide mechanisms for data authentication or freshness.

Protocol misuse: Attacks may be possible by abusing one protocol specification [186]. First, a LIN subnet can be disabled by sending a malicious sleep frame. Then, a Denial of Service (DoS) attack can be launched on a CAN bus by misusing the priority-based arbitration of the bus and sending messages only with the highest priority. Finally, a misuse of error messages can also be leveraged to disconnect CAN and FlexRay nodes from their bus.

Poor protocol implementation: The protocol implementation may not reflect the actual standard specifications. For safety reasons, the engine control ECU cannot supposedly be put in reprogramming mode when the car moves. However, real-world tests showed the inverse and actually were able in this way to isolate the ECU from any CAN communication [102].

Weak ECU authentication: It has been shown that some ECU implementations provide no or weak authentication schemes, that are easy to break by brute-force attack. Such flaws may allow to request the “protected” reprogramming mode of the ECU enabling its re-flashing and for example to create a gateway forwarding packets between two subnets [102].

Poor ECU implementation: Today's car contains up to 100 millions of lines of code [34]. This amount makes a thorough code verification impossible and may result in multiple vulnerabilities leading to buffer overflow exploits. The infotainment system of several cars was shown to present such weak spots. An input path of the Windows Media Audio (WMA) parser, not specifying the maximum input length could be used to launch a buffer overflow attack. The shellcode burnt on a CD, which is then played in the car CD/DVD player, allowed to send multiple CAN packets over the on-board network [35].

Gateway misuse: Attacks through the OBD-II port can also allow to leak (private) information exchanged over the on-board network. The manipulation of the diagnostic protocol and a replay of modified diagnostic sessions allow to make the central gateway forward both diagnostic and non-diagnostic-related traffic to the outside [77].

2.1.2 C2X communications

C2X communications include all message exchanges performed by the car with an external entity, e. g., CE devices, Internet, RSUs, other cars. A modern car leverages today a plethora of C2X interfaces and antennae, which are located on a dedicated ECU like the Global Positioning System (GPS) or directly integrated as a part of a bigger ECU, like the Bluetooth interface of the HU. Unlike on-board communications which seemed

protected by the physical car body, the security threats induced by communicating with the outside over external and often wireless untrusted network are obvious. As a matter of fact, car manufacturers always provided layers of security for these use cases, which nonetheless are not flawless. These communications can be classified depending on their range of efficiency: physical-range, short-range and long-range. The remaining of this subsection does not provide an exhaustive list of C2X communication channels, but rather focuses on the ones considered by this work, i. e., related to an IP-based context, and on related security considerations.

2.1.2.1 Physical-range C2X communications

These communications usually involve the physical tethering of a device to the car, e. g., through a Universal Serial Bus (USB) interface, the OBD-II Port, or by inserting an information medium into the car, e. g., a CD. The security risks and attacks here have already been discussed in the previous subsection and shown that these communications are currently the most investigated attack vector.

The security layer of the USB interface allows some applications developed by the car manufacturer and installed on the plugged device to establish secure communication channels with the HU. These legit applications are equipped with a certificate signed by the car manufacturer which defines their rights for a set of infotainment and car-status functions. These certificates are generally not protected and thus can be extracted from their applications in order to be misused.

2.1.2.2 Short-range C2X communications

These communications are occurring within the car cockpit and are performed by wireless-enabled devices, e. g., CE device over Wi-Fi, Bluetooth, or by on-board wireless sensors, e. g., Tire Pressure Monitoring Sensor (TPMS) via radio frequency.

Wi-Fi and Bluetooth provide standard methods allowing strong communication encryption and password/PIN-based authentication. The related antennae are usually present on the HU, which enforces an Access Control List (ACL) defining the function availability. These authorized features are generally limited and not security-critical, e. g., phone book, phone call, audio system, car statistics. Some Bluetooth interfaces showed to contain weakly protected function like `strcpy()` allowing attackers to perform a buffer overflow attack and execute arbitrary code on the HU [35].

Then, TPMSs are supposed to only communicate with the tire pressure monitoring ECU. While the car body actually shields these communications, the wireless signal was shown to be heard up to 40 meters away, which makes remote car tracking based on the sensor ID possible. Due to a lack of security on the communication protocol, it was also demonstrated that an attacker could spoof the wireless communication. The attacker was therefore able to send wrong information to the ECU, display them on the digital instrument panel and drain the sensor battery [152].

2.1.2.3 Long-range C2X communications

These wireless communications occur with entities outside the car, e.g., via IEEE 802.11p for communicating with other cars or GSM/3G for Internet or the emergency call purpose.

The security related to the GSM/3G communications is mostly under the network provider's responsibility. The car is in charge of the application-level security, e.g., build a TLS-protected channel for an IP traffic. For the moment, communications with a CE device or with Internet are routed through a back-end server acting like a firewall. The server delivers to the car only authorized and valid function calls or provides static web pages, i.e., without any embedded JavaScript code. Via these methods, critical functions like door locking or shutting down the engine are possible and relatively safe to perform. However this solution is expensive, not scalable on the long term and may not be fully secure [119].

Like the Bluetooth interface, the emergency call interface may also be vulnerable. After a reverse-engineering of the aqLink protocol, it was found that some cars were assuming to encounter traffic with a packet size of less than 100 bytes. Without any length checking procedure, a buffer overflow could be performed and make execute arbitrary code on the telematics platform hosting the aqLink interface [35].

2.1.3 Security Research

In contrast to a few years ago, today's car manufacturer cannot ignore the IT security threats anymore. As a consequence, during the last decade, both academia and industry proposed several research works and large-scale projects to secure automotive communications infrastructures.

2.1.3.1 Academic Work

The first task was to address, quantify and classify the security problems. For this purpose several approaches were taken. At a theoretical level, Wolf et al. [186] investigated the weaknesses of the on-board protocols and potential attacks. Lang et al. [106] discussed the risk of opening the on-board network to external IP-based networks and formalized a set of several attack scenarios. At a practical level, Hoppe et al. [75] tested attacks on the CAN bus and on real hardware sets on experiment tables. They also investigated the possibility of performing sniffing and replay attacks via simulation and proposed an adapted version of the CERT taxonomy [78] for the automotive purpose [72]. At a "real-world" level, the *center for automotive embedded systems security* demonstrated the feasibility of attacking the whole car. Using techniques for packet sniffing, fuzzing and reverse engineering, they compromised several cars first with a local access to the OBD-II port [102] and then remotely by exploiting vulnerabilities of external communications interfaces [35]. The rest of this section presents a selection of

new security features proposed to secure the on-board network:

Securing the CAN protocol and the on-board architecture: A first approach is to directly work on the CAN protocols. Chávez et al. [39] proposed to encrypt the CAN frame by using a lightweight RC4 algorithm and provided the related pseudocode. The authors evaluated the induced latency for different lengths of frames (greater or equal to 8 bytes) but did not consider the authentication schemes or any mechanism to establish the session key. Regarding the authentication, Nilson et al. [137] proposed to use a Cipher-Block Chaining MAC. A 64-bit MAC is produced out of 4 consecutive CAN messages, each of them receives 16 bits of the MAC in their cyclic-redundancy-code-field. But such a protocol induces delay for verifying integrity and authentication, since all messages need to have arrived. In addition it does not consider the data freshness nor the case where the MAC verification fails.

A second approach is to restructure the on-board architecture at a deeper level. Groll et al. [65] proposed to regroup the ECUs in trusted groups, where the ECUs share a same symmetric key. A key distribution *centre* in the vehicle is responsible for the key and cryptographic management. The trusted groups are defined in ACLs signed by the car manufacturer and stored/enforced by the *centre*. No real-world implementation is provided, only an evaluation of the latency induced by the encryption of the packet with several symmetric or asymmetric algorithms.

A third approach is to make use of attestation-based security leveraging the Trusted Platform Module (TPM) [169] capacities of an ECU. Oguma et al. [139] proposed to use a remote-attestation protocol to verify the validity of the software running on an ECU. A central master ECU collects the attestation hash of each ECU and generates the symmetric encryption key for the valid pairs. However, again no real-world implementation was provided.

These approaches focus on providing security (i. e., encryption and/or authentication/integrity) to the CAN protocols. However the CAN bus only provides very constrained properties that make the security management very unpractical:

- **Message length:** the CAN frames are 8 bytes long and cannot be extended. As a consequence, for 32 bits of data, CAN can only carry a MAC of 32 bits at most. CAN FD proposes to transmit frames of up to 64 bits on a CAN bus. However its adoption by the car industry is not clear and cannot yet be considered as an option.
- **Available resources:** in addition to the bandwidth, the number of CAN-IDs is also limited and cannot be extensively used to provide additional messages for key exchanges or authentication mechanisms.
- **Bidirectional protocols:** the broadcasting and event-driven nature of CAN limit the use of bidirectional security protocols, especially the use of authentication challenges for time-critical situations.
- **Real-time capabilities:** ECUs may have to respond within 1 ms [139]. In case of a protocol using a MAC, the MAC needs to be generated, transmitted and verified

within this time frame. Such delay cannot be guaranteed if the MAC is complex to compute or transmitted over several messages.

- **Interoperability:** security-enabled CAN nodes may have to be interoperable with non-security equipped nodes in order to allow an implementation of the current on-board network of a car manufacturer. As a consequence, all messages need to still be broadcasted and available to the whole network, which clearly limits the use of encryption.

In addition, none of the works previously mentioned, which focus on legacy communication buses, do really consider, if they need it, the key distribution process in itself, i. e., which algorithms and protocols are used to exchange or establish the encryption keys. They assume it flexible enough and secure. However until now, no flexible enough car-wide solution was proposed. Unlike the automotive world, the IP world already proposes for this Public Key Infrastructure (PKI)-based solutions that could be adapted for cars [55].

Developing automotive Intrusion Detection Systems (IDSs): Larson et al. [108] proposed and evaluated a CAN-based IDS. Considering that the CAN protocols are not specifying the identity of the message sender and receiver, they choose to not develop a network-based IDS, but rather an introspection-based IDS. The IDS is implemented directly on the CAN controller and inspects all incoming and outgoing packets to checks if some requirements on the message fields, the retrieval and emission rates are respected. They consider that the most critical assets to protect were the gateway ECUs, which are also the most difficult to protect per IDS since all their interfaces have to cooperate together, e. g., to detect lost or corrupted messages.

On the other side, Hoppe et al. [76] opted for a anomaly- and network-based IDS. By looking at the rate of a specific type of packet, the IDS can compare the resulting value to the normal one and detect anomaly, e. g., an increase of traffic when the car is idle. They then investigated the way of displaying the warning of an intrusion [73]. Instead of sending it to a centralized server, they chose to directly inform the driver via a human computer interface. Depending on the criticality of the attack, audio, visual or haptic messages are transmitted to the driver.

2.1.3.2 Automotive & Security Projects

During the last decade, academia, car manufacturers and their subcontractors have launched a series of large scale projects with governmental support aiming at securing cars. The European Union (EU) project SeVeCom [159] was one of the first and addressed the security issues of future vehicle communication networks, i. e., the security of C2X communications. While having designed several C2X protocols using encryption and authentication mechanisms, they mostly kept their work at a theoretical level. The field operational tests were later performed by the German project sim^{TD} [163], which implemented the protocols on their C2X communication platforms. This project also

proposed a security architecture relying on a PKI for long-term car identity as well as short-term identities in order to provide pseudonymity [18]. However none of these two projects really formalized the transition of data between outside and inside or considered the damages that external data could cause on the inside. Their on-board security was relying on strong security components on the edge of the in-vehicle network and performing the enforcement of static ACLs.

Regarding the in-vehicle security the EU project EVITA [57] designed on-board protocols and architectures aiming at providing security and trust already within the vehicle, i. e., at the ECU and sensor level. They proposed a modular framework for the ECUs, which allows to establish internal secure communication channels and leveraging embedded Hardware Secure Modules (HSMs). Some security nodes called *Security Masters* take care of the key distribution, policy management and IDS and support the other ECUs to secure their on-board traffic. As follow-up the German project “Sicherheit in Eingebetteten IP-basierten Systemen (SEIS)” investigated the feasibility of using Ethernet and IP as standards for automotive on-board communications [63]. Security was one of their major concerns. Their security architecture is partly inspired by EVITA. Part of the research explained in Chapter 3 was led during the SEIS project and concerns a secure middleware architecture. The EU project OVERSEE [143] investigated the possibility to reduce the total number of ECUs by developing super-powered ECUs fulfilling the function of several ECUs. These super ECUs leverage the HSM of EVITA and virtualize a full embedded IP network allowing communications between the embedded virtual ECUs. The German project Aramis [7] is currently running and is about designing a secure embedded multicore architecture running safety-critical functionalities. Such system would for example allow to have simultaneously on the HU a safety partition and an Android-based partition for infotainment sharing the same physical platform, e. g., the HU hardware but also the display screen.

2.2 About Tomorrow’s Car

Future automotive uses cases for driver assistance, infotainment and C2X connectivity have all increasing requirements for bandwidth availability and computation power. However, as seen in the previous section, security concerns and technology limitations in term of bandwidth and interoperability currently slow down their development and integration into cars. Part of the solution may lie in the use of fast buses and more flexible networking protocols like Ethernet/IP [157], an option already investigated by the SEIS project [63] for several reasons:

- **Limited cost:** instead of equipping each car with a model-specific and complex cable network, car manufacturer will wire each ECUs with a simpler Ethernet based network composed of inexpensive single pair unshielded cables [140].
- **Bigger bandwidth:** the automotive variant of the 100 Mbit Ethernet is a valid alternative to the MOST150 and may soon lead to its Gbit version. In addition to

a high bandwidth, Ethernet also provides a high throughput, a relevant evaluation criterion for CAN-based use cases.

- **Scalable and easy ECU coupling:** the use of automotive switches will considerably simplify the network addressing and ECU coupling. Ethernet/IP subnets will be simply plugged together via switches and routers and will avoid the configuration of complex interface gateways.
- **Available standards:** a plethora of standard protocols designed for Internet will be directly applicable or customized for the automotive purpose. First at a functional level, Ethernet/IP already benefits from efficient transport (e.g., TCP, UDP) and network management (e.g., ICMP, ARP) protocols. Then at a security level, they provide mature and secure protocols already strengthened against real-world attacks (e.g., MACsec, IPsec, SSL/TLS).
- **Easy migration strategy and flexible communication paradigm:** Ethernet and IP allow unicast but also broadcast and multicast communications, a necessary requirement for some CAN-based use cases. Ethernet also offers synchronous and isochronous data transmission (IEEE AVB [85]), as MOST does it for Audio/Video traffic. Finally an Ethernet/IP-based API can easily be compatible and support the existing APIs of CAN-based and MOST-based applications.

While being functionally suitable Ethernet/IP does not directly solve all security issues, e.g., security communication management, function/data access control. Since IP and Ethernet are well-known standards they could actually lead to more attacks on the on-board network. The rest of this section discusses in more details the following points: (1) the architecture of the future on-board network in Section 2.2.1, (2) the future on-board communication protocols in Section 2.2.2, (3) the future on-board security protocols in Section 2.2.3 and (4) the future C2X multi-platform antennae in Section 2.2.4.

2.2.1 The Future On-board Network

Figure 2.2 presents a simplified architecture of a future on-board network. In a similar way as today, the future one will be composed of several Ethernet/IP subnets interlinked at the level of a central router. Each ECU of each subnet will be connected via a hierarchical tree, composed of several switches. This architecture allows to reach an optimal tradeoff for suitable performance and Quality of Service (QoS), i.e., to balance the tree and not have a congested subnet [109]. Figure 2.2 only shows the first level of each tree. Unlike current architecture where ECUs are physically organized around a domain, the ECUs will be in the future only logically assigned to a domain and will have to rely on security for a domain-based separation. Their physical assignment to a specific subnet will depend on their localization of the car and on an optimization of the whole tree. Additionally, all wireless C2X interfaces will be regrouped on a Multi-Platform Antenna (MPA), described in more details in Section 2.2.4. Both OBD-II port and MPA will be directly connected to the central router via the proxy. The rest of this

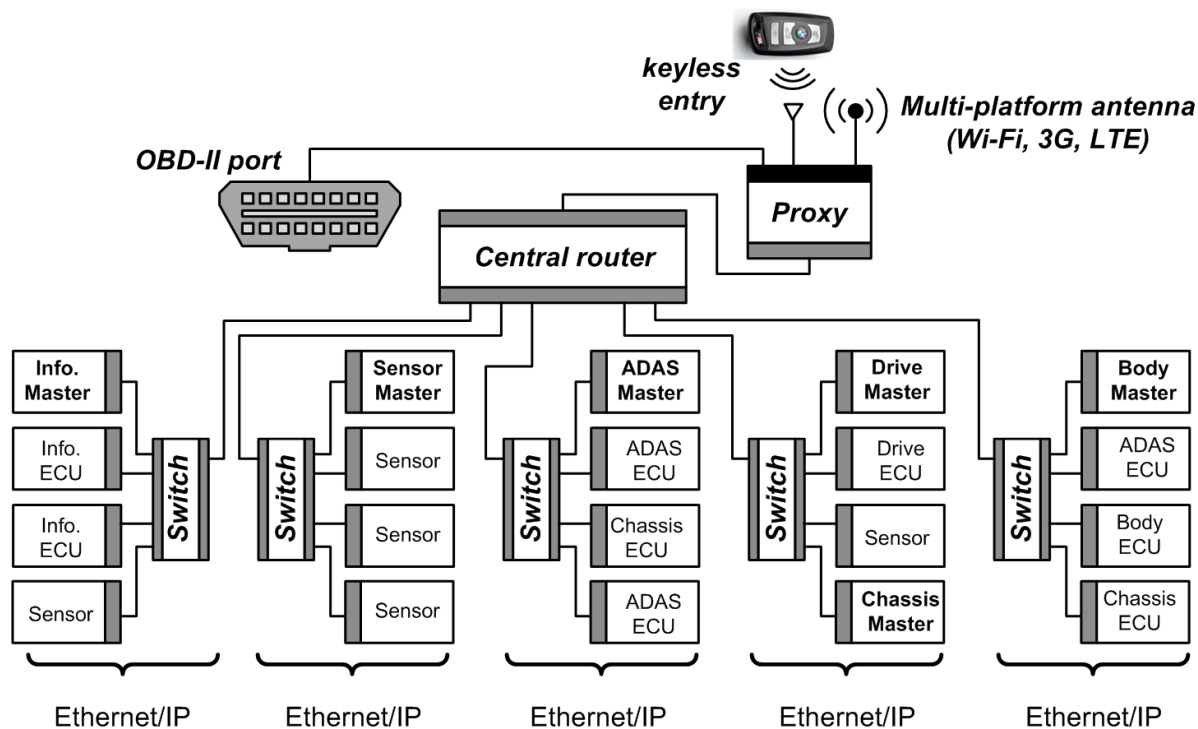


Figure 2.2: Schematic of a future on-board network. The ECUs (□) are connected per Ethernet buses. The proxy (□) is connected to the central router (□) and is the only component managing C2X interfaces such as the OBD-II port or the multi-platform antenna. This distribution of the domain-based ECUs and master is given as an example. (Info. = Infotainment)

section presents the different domains, namely the Drive-, Chassis-, Body-, Advanced Driver Assistance and Safety (ADAS)- and Infotainment-domain.

The *Drive domain* concerns the ECUs responsible for the engine and transmission control. Both communications between engine ECUs and between engine- and transmission ECUs can be considered as real-time communications with high reliability requirements.

The *Chassis domain* contains the ECUs responsible for driving control use cases like the Electronic Stability Control (ESC) or the Anti-lock Braking System (ABS). They also involve real-time communications and need to react based on different sensor values in a short time to ensure their safety purpose.

The *Body domain* hosts the ECUs managing several comfort functions like the electric seat adjustment or the control of the convertible roof. These functions are not as critical as the ones of the two previous domains. But their malfunctioning could still cause severe driving discomfort or financial car damages. Additionally, the diagnosis functionality is usually part of this domain.

Since a decade, numerous ADAS systems have been implemented in cars, e. g., adaptive headlights, automated parking, adaptive cruise control, lane departure warning and

night vision. In order to achieve their purpose, the use cases of the *ADAS domain* use complex sensors based on ultrasounds, radars, lasers or cameras. Additionally, they require to have a large amount of data transported from the sensor to the ADAS ECU and are subject to real-time requirements comparable to the Chassis or Drive domains.

The *Infotainment domain* includes the ECUs taking part in audio/video use cases as well as some C2X connections, e. g., for the Internet. It concerns, among others, the HU with originally the FM radio and later the management of additional media data like CD/DVD chargers, digital/satellite radio and digital TV. Another use case is the on-board navigation, which relies on the current GPS position, pre-stored maps, sensor information and additional data received via radio waves or Internet. Most of these use cases are enabled via an Internet connectivity, which allows numerous services, e. g., mails, weather consultation, exchange of information between vehicles. It may also soon enable to perform a remote car diagnosis, in order to optimize the maintenance and to know in advance which components to fix in the garage.

With an increasing number of ADAS features, it now makes sense to change in depth the car structure. Instead of having each costly sensor directly coupled to an ADAS ECU, it becomes interesting to decouple it from its ECU. A dedicated ECU aggregates all sensor information into an environment model that can be sent to all interested ECUs. Because not all information of this model may be interesting for a resource-limited ECU, an intermediary ECU in each domain, called master ECU receives the full model and redistributes the relevant information. In a realistic vehicle, a domain will contain at least a master ECU and potentially some other ECUs depending on which options and services were ordered by the customer.

Thus the *Sensor domain* contains all the sensors of the car. The sensor master collects all sensor information, generates the environment model and periodically forwards it to the other masters.

2.2.2 The Future On-board Communication Protocols

Before specifying the protocols that the on-board network will use, it makes sense to briefly define the different communication streams that have to be carried. These streams can be classified as following and will be described in more details along this section:

- Automotive infrastructure management;
- Transport of Audio/Video (A/V) data;
- Clock synchronization;
- Control traffic;
- IP-based Data Communication.

The considered protocols are presented in Figure 2.3. These protocols may be categorized in 5 clusters: 2 clusters as basis of the communication stack system, namely the

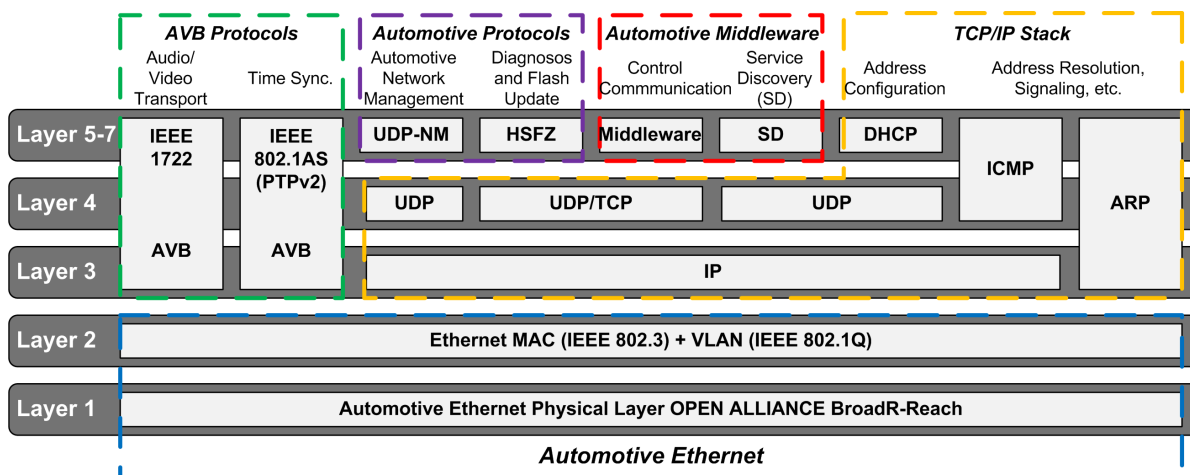


Figure 2.3: Automotive protocol suite (adapted from [157]). These protocols are presented in function of their disposition in the ISO/OSI model. The dashed boxes designate the categorization described in section 2.2.2. In this figure MAC means media access control.

Automotive Ethernet and the *IP/TCP Stack*, and 3 clusters transporting the communication streams mentioned in the previous list, namely the *Automotive Protocols*, the *Audio Video Bridging (AVB) Protocols* and the *Automotive Middleware*.

Basis clusters: As basis of the on-board communication stack lies the protocols for the *Automotive Ethernet*. It is composed of the BroadR-Reach physical protocols standardized by the OPEN ALLIANCE that will serve to encode the Ethernet packets over single-pair unshielded wire. Then on top of the physical layer, the layer-2 protocols compliant with the IEEE standards 802.3 and 802.1Q provides the mechanisms for media access control addressing and Virtual Local Area Network (VLAN).

The *IP/Transport Control Protocol (TCP) Stack* protocols sit above the *Automotive Ethernet* cluster. They provide mechanisms for IP addressing as well as several other features for address configuration, resolution and signaling. This cluster also provides the transport protocols for all application-layer protocols of the car. This cluster is not automotive-specific and comes into car without any major modification.

Stream clusters: The management of the automotive infrastructure is provided by two car-specific protocols relying on the IP/TCP stack. First, the UDP-NM protocol is specified by the AUTOSAR standard committee and specifies the protocols for coordinating the transition between normal functioning and bus-sleep mode of the network [9]. Then, the second protocol concerns the flashing and diagnosis of the ECUs, e. g., the proprietary High Speed Fahrzeugzugang (HSFZ) protocol of BMW, and may rely either on User Datagram Protocol (UDP) or TCP depending on the situation requirements.

Then, unlike MOST which was designed for the *A/V transport*, Ethernet is more multi-purpose. The AVB Task Group designed protocols for transporting A/V streams

in local networks and defined IEEE 1722 standards running directly on the Ethernet layer [86]. This standard achieves a very precise and synchronous playback when used with a clock synchronization protocol like the standard IEEE 802.1AS [84]. In addition the IEEE 1722 allows to save the overhead caused by the IP and transport headers.

Clock synchronization is also essential when synchronizing the speaker playback and video display. Thanks to hardware-based timestamps, the IEEE 802.1AS reaches a high precision, i. e., under the microsecond range. This standard is an extension of the precision time protocol version 2 (PTPv2) and can be used directly on the Ethernet layer or over UDP/IP. The precision is obviously the highest with the version running directly on Ethernet. However, the high precision clock may impact the security. The hardware-based timestamping is performed by the Ethernet device after the security mechanisms and as a consequence the integrity and authentication of the frame may seem corrupted.

Then, in addition to these features, the car needs some *Control Traffic* mechanisms, e. g., to start/stop an A/V stream, to get the current speed, GPS coordinates, to transfer the driver's playlist information. Due to the diversity of tasks and requirements, the car needs flexible and general protocols. These tasks are therefore assigned to the middleware and its associated Service Discovery (SD) mechanisms. The description of this layer and its security constitute the main focus here and are addressed in more detail in Chapter 3, 4 and 5.

Finally, the *IP-based Data Communication* concerns the infotainment domain but are not depicted in Figure 2.3. The most explicit example considers the network file system to establish between HU and RSE, so that the RSE can access the navigation information or the movies stored on the HU. Several protocols already exist for this purpose, perform over IP and UDP and propose their own security mechanisms, e. g., FTP [147], NFS [162].

2.2.3 Securing the Future On-board Communication Protocols

This section aims at selecting the most suitable security protocol to secure the on-board communications. Therefore based on the type of streams defined earlier, several requirements for the communication security can be defined:

- *Protection of A/V content*: for legal reasons, the protection of such contents may be required, in order to prevent an attacker from eavesdropping the content and reusing it illegally.
- *Protection against unauthorized manipulation*: it concerns the protections against an attacker, who could intercept messages, modify them and reinject them or just replay them at a later time. These manipulation aims at gaining additional unauthorized features or disabling safety mechanisms. They could damage the car integrity and also lead to warranty fraud.

Table 2.2: Comparison of different protocols for securing the on-board communications.

Evaluation Criteria	MACsec [83]	IPsec [99]	SSL/TLS [48] - DTLS [150]
Protected layers	L2 and above	L3 and above	L5 and above
Protection paradigm	one-hop	end-to-end	end-to-end
Application control	No	Yes with [185]	Yes
Communication aggregation	Yes	Yes	No
Key exchanges available	Yes	Yes	Yes
Support preshared keys	Yes	Yes	Yes
Support EAP [2]	Yes	Yes	No
Authentication	Yes	Yes	Yes
Encryption	Yes	Yes	Yes
Robustness against Injection	Yes	Yes	No
Protect AVB protocols	Yes	No	No
Protect control data	Yes	Yes	Partially
Protect IP communications	Yes	Yes	No

- *Privacy protection:* privacy is a growing concern. Due to the increasing number of customization and location-based services, the car processes a considerable amount of sensitive data, e. g., current position, address book, data from social networks.

With Ethernet and IP, the security risk increases: the attacks are easier to perform since they are well-know technologies. Additional communication security mechanisms are required. As earlier mentioned, the focus here is to asses which security protocols are suitable for the defined use cases. A more detailed attacker model is defined later in Section 2.3 and considered to assess the main security concepts of this thesis.

Three mature security protocols hardened by the “public community” and operating at different levels of the communication stack have been selected and are evaluated in Table 2.2.

Security protocols: MACsec [83] protects the communication stack above the Ethernet layer (L2) and is thus very powerful. However this protocol usually requires a hardware-supported encryption, which may be a big disadvantage for in-car embedded systems and their resource-limited platforms. This protocol also does not provide end-to-end security and only protect communications between two IP nodes directly connected.

Internet Protocol security (IPsec) [99] can be used in tunnel and transport mode and protects the layers 3 and above. Considering the staticity of the on-board network, IPsec channels can be opened between all required ECUs. Therefore only the transport mode is considered here. Like MACsec, this protocol can allow several application

communications to use a same secure channel (i. e., communication aggregation).

Secure Socket Layer (SSL)/Transport Layer Security (TLS) [48] and Datagram Transport Layer Security (DTLS) [150] only protect the layers 5 and above. The application has to embed the security mechanisms in its code and can only open a secure channel for itself. These protocols allow to control the security from the application level, which is interesting in Internet but relatively limited in a closed environment like the car. Besides, SSL/TLS provides a limited robustness against TCP injection [173].

SSL/TLS and DTLS do not provide enough security guarantees against injection and do not protect all protocols of Section 2.2.2. Unlike IPsec, MACsec may protect the A/V traffic and the clock synchronization. However, certain content providers may require to use High-bandwidth Digital Content Protection (HDCP) [113] and make the use of MACsec less relevant. Since A/V traffic is not critical, HDCP-like security mechanisms should suffice. Then the clock synchronization of infotainment use case, e. g., synchronization of audio and video, is also not critical. Even if the drift between audio and video is too big, the discomfort cannot be a cause of disturbance for the driver. Therefore such synchronization may be unprotected. However, some critical ECUs may also require date and time information, e. g., feature (de-)activation. For these cases, a high precision is little relevant, therefore the synchronization can be performed with the IEEE 802.1AS standard running over IP and UDP. The synchronization can take reliable time sources like the GPS or a trusted back-end server and transport the traffic to protect over IPsec. Other traffics, like for control or IP-based data communication, can also be protected with IPsec from the emitting ECU to the receiving ECU and over the same security channel. The use of IPsec is therefore recommended for control, IP-based data communication and time synchronization with non-high precision.

Key exchange: In order to establish IPsec channels, two security key managements are possible: (1) statically configured keys or (2) dynamically exchanged keys through the on-board network. Usually IPsec uses the Internet Key Exchange version 2 (IKEv2) for dynamic key establishment [97]. However its complexity may be problematic for embedded systems like in the car. Therefore a solution would be to use static key stored in a secure part of the hardware or to use a lightweight version of IKEv2 [100].

Protocol Robustness: Considering the high safety requirement of the car, car manufacturers have to make sure that the different communications do not influence each other, e. g., video streams should not disturb the traffic received by the engine controller ECU. The domain-based architecture is essential here; the domains are isolated from each other through VLAN techniques. The priority of a domain traffic can be balanced according to its importance. In addition, the domain masters, who benefit from additional computation power, may be used as security nodes, isolating the domain from each other and even being able to detect a compromised node, like an IDS.

In order to increase the system robustness, the port-ingress rate limitation provided by default by the Ethernet switches can be leveraged. The on-board communication patterns are quite static and known on beforehand by the car manufacturers. Therefore this feature could easily limit the propagation of DoS attacks for example.

Adding security at the communication level obviously limits the possibility of an attacker to add a new active component to the on-board network, replace an old one and to listen/inject messages. However, as seen in Section 2.1, most attacks were performed by getting plugged to the OBD-II port and just leveraging this “ready-to-use” interface. Already today, in high-end cars, the OBD-II port include a CAN-Ethernet port and could be protected by IEEE 802.1X security [87]. This standard supports secure service identification as well as some Layer-2 encryption. But due to some governmental regulations, the car has to provide an easily accessible port allowing to access a few internal functionalities without security. Even if limited in numbers, these few functions provide a direct on-board access and could still be leveraged to launch a DoS attack.

As a conclusion, even if IPsec, a lightweight IKEv2 and some switch filters provide security at the communication level, they do not specify what should happen on the ECU and its applications during the communication channel establishment and access control enforcement phases. With this in mind, Chapters 3 and 4 aim at answering these shortcomings.

2.2.4 The Future Multi-platform Antenna

Research to design new MPAs gained a strong interest with the development of intelligent transport vehicles. Such vehicles require C2X connectivity with both other vehicles (C2C) and the surrounding infrastructures (C2I). Future use cases foresee that all vehicles will gather sensor information about the traffic and the road state, aggregate them and share them with other users, i. e., via the user’s devices, the infrastructures or other cars. The considered cases mostly aim at improving the safety, e. g., collision avoidance, emergency braking, or hazardous location notification, but not only, they may also soon concern the infotainment domain. For these purposes, the cars need reliable low-latency vehicular communications. The combination of multiple antennae, i. e., with IEEE 802.11p at 5,9 GHz for C2C and with the 2G/3G/4G cellular standards for C2I, improves the system reliability and the robustness of C2X communications [121].

Due to aesthetic reasons and cost considerations, car manufacturers restrict the number of mounting points and locations of these antennae. By current conventions, the signals of the different antennae of the car are processed centrally on the HU and represent a significant cost of cabling. A cheaper approach consists of mounting the MPA at the roof top location and coupling it to a centralized gateway performing all the signal processing and redistribution of the C2X traffic to the on-board network. The MPA and processing unit are as consequence a whole ECU, which is called communication proxy for the rest of this thesis.

In addition to the mentioned long-range C2X interfaces, the proxy can also deal with short-range interfaces like Bluetooth or Wi-Fi to couple the driver’s smartphone and may even be in charge of the OBD-II interface. The security of the communication leaving the on-board network over the MPA is ensured by existing standard solutions, i. e., Bluetooth PIN/encryption security, WPA2 for Wi-Fi, signature-based security for

C2C communications. On the other hand, the security of cellular communications (i. e., 2G, 3G, 4G) is still ensured by the network provider. Although these communications may be integrity-protected, authenticated and/or encrypted, such security mechanisms do not consider the authorization problematic, i. e., which messages are authorized to get in or out. With such a communication proxy, car manufacturers have now the possibility to set up a central security C2X gateway enforcing a consistent security model over all external communication interfaces. The Chapters 3 and 4 aim at designing and developing such a C2X access control model.

2.3 Security Threats and Risk Analysis

The car was not developed as a “security-by-design” product. As a consequence, the surface of attack is quite broad. This section describes first the traditional attackers and threats of the automotive world and then focuses on the ones considered by this work. The section is organized as follows: (1) presentation of the attacker profiles in Section 2.3.1, (2) description of their motivations in Section 2.3.2, (3) analysis of the exploitable threats in Section 2.3.3, and (4) formal definition of the attacker model for this thesis in Section 2.3.4.

2.3.1 The Attackers

Every person in contact with the car, physically or remotely, during its production or during runtime, may be considered as a potential attacker. The attackers as well as their capacities are listed as follows:

- *Owner/driver* is a legal user of the car. She is not usually capable of conducting extensive modifications on the car. She entrusts others (e.g., the tuner) for performing them. She benefits from the car manipulations or attacks (e.g., opening the convertible roof while driving), but may not be aware of the consequences, e. g., loss of functionality, car degradation, warranty/insurance issues.
- *Motor mechanic* is usually mandated by the owner to perform the maintenance and reparation. Through this context she is capable to manipulate the car.
- *Tuner* is entrusted by the owner to perform modifications on the car, e. g., component installation and/or configuration increasing the car performance.
- *Employee of the car manufacturer* can, as an insider, add or manipulate, e. g., during implementation phase, some functionalities that could be used later during an attack. She can also pilfer some car components and sell them later.
- *Hacker* is an IT expert interested in the functionalities of the cars and investigating their security weaknesses. She usually uses reverse-engineering techniques and aims at discovering weak/flawed interfaces. She is usually more interested in publishing her work than causing any real damage and earning money out of it.

- *Organized crime* aims at taking illegally possession of the car, i. e., stealing and exploiting it . Such organization benefits from the support of IT specialist familiar with all kind of attacks. Their resources in term of money, skills and people are limitless.
- *Terrorist* uses the car for its illegal activities. For this purpose, she also aims at being as little conspicuous as possible, when targeting a car. Like the organized crime, her resources may be limitless.

2.3.2 Their Motivations

Before choosing which attacker to focus on, it is important to really understand which motivations drive an attacker. After listing the main motivations to attack a car, Table 2.3 proposes to link the attackers to their motivations:

1. *Car theft*: the car is stolen with its entry key or by thwarting the door locking and immobilizer system, in order to sell the car as spare parts or as a whole.
2. *Car tuning*: this action is usually motivated by the owner, who wants to add more value to his car but without having to pay for it. However the owner is usually not capable of performing such modifications (e. g., performance increase, functionality activation) and asks the tuner to do so. Organized crime may also use car tuning to increase the selling price of the car.
3. *Illegal usage of stolen electronic equipment*: The owner is interested in paying less for a spare part. Like in the previous case, the owner is incapable of performing the installation and asks the tuner to do it. The motor mechanics can also use these cheaper parts to fix his customers' cars and charge them with the price of a legit component.
4. *Blackmail*: An employee of the car manufacturer can threaten her employer to reveal some production secrets or to lower the quality of some newly produced cars. A hacker can threaten the car manufacturer to perform attacks on its cars. The organized crime can blackmail a person by taking her car or using personal information extracted from her car. The terrorist can threaten to manipulate a large amount of cars in order to ask for the release of a prisoner or for other forms of support.
5. *Theft of intellectual property*: An employee of the car manufacturer can steal intellectual property of her company to sell it or blackmail the company. In both cases, it is motivated by a large financial gain. The organized crime can also do it as part of its usual activities. Information about critical functionalities like the car locking, immobilizer or other protections are very valuable.
6. *Manipulation of displayed information*: The owner can mandate a tuner to lower the mileage of the car, in order to increase the car selling price. The organized crime may have the same interest but also with other car information, e. g., wrong traffic information to reroute a car of interest.

7. *Leverage of the emergency mode*: The emergency mode is a error-safe mode launched after a critical error of the electronic of the car, which restrains the car capacity, e. g., maximal speed, engine rotation. A car manufacturer employee can leverage the threat of this mode to blackmail her employer. A hacker can attempt to launch this mode for a research purpose. A terrorist can launch a large-scale attack starting this mode and propagate as much damage and panic as possible.
8. *Increase of the attention on security issues*: A hacker can be interested in showing the feasibility of exploiting an interface weakness. Such a goal is usually followed by the academia or other conference about IT security [19].
9. *Increase of her own reputation*: A hacker can through a successful attack increase her reputation. A terrorist could also use an attack to increase the media coverage for her cause.
10. *Passenger endangering*: This motivation mostly concerns the terrorists, who can through this mean achieve their goals and get attention on their cause.
11. *Privacy infringement*: A hacker could publish online or sell some driving profiles. The organized crime could use this information to follow a car or a specific person. A terrorist could follow the location of a high-value target for an assassination attempt or a bombing.
12. *Leakage of information/secrets*: The gain of secret information mostly concerns several aspects, e. g., retrieval of specifications about the cryptographic behavior of the car (keys and protocols), of information about online financial transactions, of passwords.

These motivations can be classified in two clusters: for motivations 1 to 6, the attacker is motivated by a financial gain. On the other hand, the items 6 to 12 are motivated to show the weaknesses of the car in order to harm the car, its passenger, the car manufacturer or to benefit from media coverage. This thesis focuses on the on-board IT security and as a consequence will not bring solution to the motivations 4 and 5 which mostly concern the human factor.

2.3.3 The Threats That Can Be Leveraged

Before describing the threats that an attacker could leverage for her own purpose, the use cases considered by this work should be clearly stated. They are depicted in Figure 2.4 and consider an automotive on-board network as it has been presented in Section 2.2. They feature both internal and external untrusted components, over which the car manufacturer has no control. Internally, the TPA benefits from the computation power of the HU. The TPAs have to be opposed to the original software, which is present on the ECUs, installed during the car assembly and developed by the car manufacturer or its subcontractors. The TPA may communicate with several ECUs applications and with the Internet and some CE devices over the MPA. In addition, Internet services and CE

Table 2.3: The attackers and their motivations. (The attacker and motivation categories have been shortened, but still follow the indexing presented in this section.)

Motivation	Attackers	Owner/ Driver	Mecha- nics	Tuner	Emplo- yee	Hacker	Orga. Crime	Terrorist
1) Car theft							👍/⚙️	
2) Car tuning		👍		⚙️			👍/⚙️	
3) Stolen. equip.		👍	👍/⚙️	⚙️	👍/⚙️		👍/⚙️	
4) Blackmail					👍/⚙️	👍/⚙️	👍/⚙️	👍/⚙️
5) Intel. prop.					👍/⚙️		👍/⚙️	
6) Disp. manip.		👍		⚙️		👍/⚙️	👍/⚙️	
7) Em. mode					👍/⚙️	👍/⚙️		👍/⚙️
8) Sec. issues						👍/⚙️		
9) Reputation						👍/⚙️		👍/⚙️
10) Endangering								👍/⚙️
11) Privacy						👍/⚙️	👍/⚙️	👍/⚙️
12) Info. gain			👍/⚙️		👍/⚙️	👍/⚙️	👍/⚙️	👍/⚙️

👍: motivates the attack / ⚙️: performs the attack

devices may also be authorized to get access to some on-board applications. The TPA, the CE devices and Internet services are considered as conform to the internal API of the car. However they may present a poor and unsecure implementation that may be exploited by an attacker.

The rest of this section presents the interfaces threatening the on-board communication infrastructure, namely (1) the on-board software, (2) the CE devices, (3) Internet, (4) the TPAs and (5) the OBD-II port.

Original ECU software: The computerization of the car leads to an increase of the on-board application complexity and as a consequence of the amount of code to maintain and verify. While the application specifications are provided by car manufacturers, the development is usually performed by multiple subcontractors. Even if car manufacturers assess functionally all software modules, their quality in term of security remains quite heterogeneous. As shown in Section 2.1, the car software presents multiple communication interfaces and weaknesses, which are actively investigated by the research community.

The attacks on an automotive software are very similar to the ones present on traditional distributed software systems. These attacks can be classified based on their goals:

- *Software Corruption:* Attacks aim at modifying or destroying automotive data and

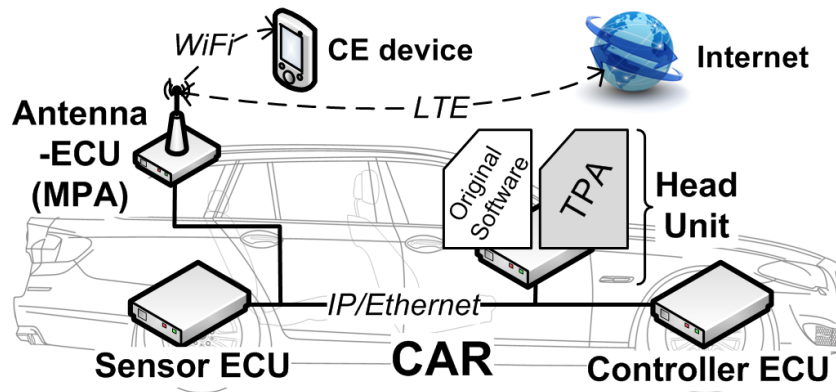


Figure 2.4: Considered use cases. Solid right-angle lines represent the wired on-board network. The dashed arrows represent external communications over different wireless networks.

services. They are usually resulting from software and implementation weaknesses, i. e., weak security mechanisms, leveraging a poor implementation, exploiting a security weakness to launch some executable code via buffer overflow [4], format string [158] or Return-Oriented Programming (ROP) [160] attacks. These attacks generally allow an unauthorized usage of the ECU, e. g., discovery of wrong services, ECU memory modification, ECU re-flashing.

- *Information Disclosure*: Successful attacks result in unauthorized access to middleware assets (mostly data). Attackers aim at stealing automotive information from the driver (privacy infringement) or from the car manufacturer (industrial espionage). Like previously, these attacks leverage the poor implementation of the authentication and authorization mechanisms of the ECU.
- *Service Interruption*: Here the service availability is affected, in order to produce delay or block the application. Attacks aim at making the service unusable or slower, they usually consist in resource exhaustion (e. g., jamming, flooding), unauthorized service deactivation or in producing errors (e. g., injection of bogus packets).

CE device: User-installable applications for CE device are already used to customize the car [80, 88]. However their integration presents a significant security risk. Even when testing the applications and controlling their publication via some official distribution channels like the Appstore of Apple, the CE industry does not manage to eliminate all malicious applications [164]. A malicious application on a CE-device could steal and use the security credentials stored on the device to establish a secure communication channel with the car. Such an application could get authenticated by the proxy, get a direct access to the on-board network and launch the software attacks mentioned in the previous paragraph. The user's security unawareness and her weak security configurations (e. g., weak password, no or misconfigured security software) increase this risk of corruption.

In order to circumvent some of these problems, mobile Operating Systems (OSs) (e. g., iOS, Android) provide libraries implementing secure communication protocols, strong authorization and isolation mechanisms, which are reliable as long as the device is well-configured and not rooted or jailbroken. For Android, academic works propose additional solutions such as taint tracking [54], virtualization [107], behavioral analysis [187], enforcement of mandatory access control [127] or analysis of remote duplicates [145]. These approaches mostly concern the internal security of the CE device and are little relevant to ensure the in-car security. Promising work about remote attestation for mobile devices has been published [133, 15], but is based on trusted hardware which is still relatively far from reaching mass production.

Internet: Internet offers a limitless number of online infotainment services. However these services and the servers, on top of which they are running, present similar risk to the ones encountered by the CE device. As soon as they get authenticated and have their messages forwarded by the proxy within the car, they may, as previously, launch a software attack.

To ensure some security on Internet, major IT security actors like Symantec [165] propose to support companies and their online services to build trustworthy services. They confirm the service identity by signing the SSL certificate of the web site and also regularly scan the servers to detect the presence of a malware. All these actors take advantage of their reputation and the fact that most current web browsers recognize them as valid certification authorities.

Another solution for trust is to leverage the experience of a community or of other Internet peers and to establish a reputation-based trust as it already exists for online services [161] or wireless ad-hoc networks [166]. However, these solutions are not security-proactive and necessitate to detect an active attack in order to propagate the information to other community members.

Third-Party Application (TPA): A last option to provide innovative infotainment services is to open cars to TPAs. The TPA could benefit from its internal status to easily get access to on-board functions while communicating with external entities like Internet or the users' CE devices. Obviously this integration raises numerous security questions, since the security risks are even higher than with the CE devices and Internet. The TPA would be directly installed on an on-board platform like the HU, which does not directly host very safety-critical applications. However the TPA could still cause driving discomfort by compromising the HU. Besides, with a direct writing access to the on-board network, the TPA could also send and receive data, making it easy to launch software attacks on remote safety-critical ECUs.

The in-car integration of TPAs is only at its beginning [60, 115]. Car manufacturers usually provide the developers with an adapted in-car API and invite them to submit their applications. Before their acceptance, the applications go through a validation process assessing their programming quality, compliance with the car as well as if they may excessively disturb the driver's focus. This approach is the one adopted by Apple and has also shown to not be flawless. Until now, no publication reported any security

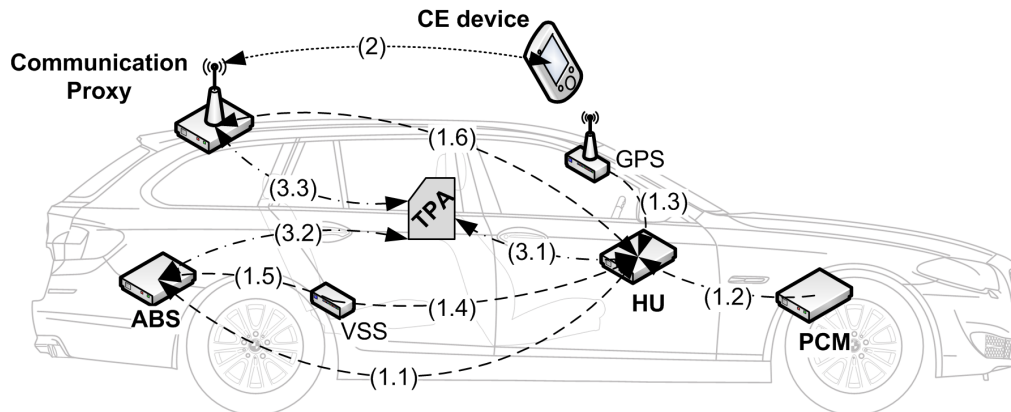


Figure 2.5: A concrete big use case. This use case features distributed components communicating together. The arrows linking on-board components are symbolic communication link and do not reflect the structure of the on-board network. The communications are numeroted and categorized whether they happen on-board (1.X), with the outside (2) or with the TPA (3.X).

issues yet and very few scientific works addressed these issues from a security point of view. Another interesting approach from Daimler proposed to integrate to the on-board network a dedicated ECU running a custom Android [144]. But even if the ECU was physically separated from any neuralgic component of the car, no real solution against on-board attacks on other ECUs was provided, e. g., DoS attacks.

OBD-II port: As explained in Section 2.1, the OBD-II port has been extensively used to attack the on-board network due to its easy access, i. e., generally located under the dashboard. Its unprotected accessibility allows to easily plug a device that can either flood the on-board network, retrieve sensitive information or send a security exploit compromising an ECU.

This thesis aims at improving the information security in cars and defines the following goals: (1) design of an on-board secure software architecture allowing the establishment of internal and external secure communication channels and performing access control for both data management and function access; (2) design of secure mechanisms for the on-board integration of internal (e. g., TPA) and external (e. g., online service) components.

An overall use case: For the purpose of discussion, a use case involving all “threats that can be leveraged” is built up in this paragraph. This *big* use case will be used later in Chapter 3, 4 and 5 to evaluate the security gains provided by the developed architecture. The use case is presented in Figure 2.5.

This use case features both on-board and external components and both original automotive and third-party developed pieces of software. The on-board entities are listed as follows:

- The HU generally referred to an ECU managing the stereo system of the car, i. e., radio, audio functions, CD player. A modern advanced HU goes a little bit further and now controls the whole infotainment system of the car, e. g., music, navigation and also some vehicular functions like the door chimes. Basically, it handles critical information like intellectual property of the car manufacturer or driver's private data. But it does not trigger safety-critical functions. A faulty HU would not have dramatic consequences but may disturb the driver.
- The ABS, contrary to the HU, triggers safety-critical functions but does not handle sensitive information. The ABS allows the wheels to maintain tractive contact with the road and prevents them from locking up while braking. ABS and HU can communicate together in order to exchange braking status information. A faulty ABS could have dramatic consequences and make the driver lose the control of her vehicle.
- The Powertrain Control Module (PCM) range of action is not as safety-critical as the ABS. The PCM gets information from various sensors in order to control some actuators of the combustion engine and ensures its optimal performance. HU and PCM communicate together, e. g., to exchange the engine status. However, PCM and ABS are not supposed to exchange any information. A faulty PCM would result in a malfunctioning engine, so in poor car performance. However the driver could still keep control of its vehicle.
- The Vehicle Speed Sensor (VSS) measures the transaxle output, i. e., wheel speed, and sends it to the PCM and the ABS. With such information, the PCM can modify some engine functions, e. g., ignition timing, transmission shift points and the ABS can optimize the brake pressure. A faulty sensor may cause some drivability issues considering the number of functions it is involved in.
- The TPMS monitors the air pressure inside the pneumatic tires and reports it to the ABS. A faulty TPMS would make the ABS less efficient and would result in a bad tyre wear out.
- The GPS module receives continuous information about the vehicle location, time and weather and provides it to the HU. A faulty GPS module may provide wrong information and decrease the quality of the navigation system.
- The communication proxy is the main gateway towards external wireless networks, e. g., Wi-Fi, 3G, Bluetooth. A faulty proxy would prevent the car from communicating with the outside.
- The TPA is a *my Driving Coach* application. It receives personal driving information (e. g., braking, engine, road information) from several ECUs, e. g., HU, ABS. It aggregates them, sends them to a CE device or to an online server and receives in return driving recommendations to display. The figure 2.5 does not reflect any aspect of the TPA integration, i. e., hosting on a traditional or dedicated ECU. The TPA should not directly communicate with the ABS or the PCM due to safety considerations.

Outside the on-board network, the CE device may communicate via the communication proxy with the TPA and the ECU A only. The CE device makes use of a legit API released by the car manufacturer. However, the communication API with the TPA cannot be regulated in a same way.

2.3.4 The Attacker Model

Adversary models are essential for security risk analysis and can also be formalized. For example, the Dolev-Yao-model [50] defines an attacker with full control over the network who can intercept, modify and replay all exchanged messages.

In this work, the adversary model is based on an extended version of the Dolev-Yao-model. The attacker may be present on both on-board and external C2X networks. She is assumed to have a timely unrestricted access to the vehicle as well as full technical knowledge of the system, i. e., of the protocols and running software. However, she is assumed to be polynomially bounded in computational power and storage. Thus the current cryptographic primitives (e. g., AES, RSA) can be assumed to be secure, since there is no algorithm known to break them in a reasonable time. Besides she cannot successfully guess random numbers. However, due to the scope of this thesis work, the attacker is limited to software-based attacks. Thus she cannot physically tamper any ECU, e. g., memory content reading or re-flashing. Hardware-based attacks are in principle out of reach of a software/middleware-based security solutions. This work partly addresses DoS attacks, but does not focus on them.

Concretely, her attack of surface includes on one side the on-board wired network including the OBD interface and on the other side all C2X communication channels able to carry IP packets, i. e., Wi-Fi, Bluetooth for short range communications and 2G/3G/4G for the long range ones. Other communication channels for key-entry, radio-channels and other addressable channels (emergency calls, remote diagnostics) are considered as out of scope. Additionally, the attacker can compromise external entities, like the driver's CE devices or a valid online service. She can also use the TPA interface, i. e., publish her own malicious TPA or compromise a TPA already installed in the car.

The attack scenarios can be generalized to 2 groups of attacks: (1) *the integrity attacks* related to the attacker motivations 1 to 3 and 6 to 10, and (2) *the confidentiality attacks* related to the motivations 8, 9, 11 and 12:

1. **Integrity Attack Scenario:** The attacker leverages her access to the on-board network to send bogus packets (e. g., a shellcode) or in case of a TPA access/modify local resources of the HU (e.g., filesystem) in order to disturb the car functioning, e. g., disabling the braking assistance, modifying the information displayed by the instrument cluster.
2. **Confidentiality Attack Scenario:** The attacker can also leverage her access to the on-board network, by trying to elude her authorizations and retrieve sensitive data, directly via the proxy or via another application from which the attacker is

able to receive messages. More specifically, thanks to a malicious TPA, the attacker can get access to sensitive data, stored on the HU (e.g., the home address of the driver in the navigation module) or received from another service (e.g., preference settings of a user from the seat controller). Even without the appropriate permission, the data may be sent to the outside by the TPA, either directly over the proxy or through an intermediate step, for example a buggy service communicating with the outside.

These two scenarios are later used in Chapter 4 to evaluate the proposed security solutions.

2.4 Automotive Functional Requirements

Automotive applications are very demanding pieces of code, subject to drastic functional requirements. The rest of this section lists some additional requirements that an automotive security architecture should fulfill:

1. *Safety considerations*: parts of the software components have high safety requirements, e. g., assistance for steering and braking. The software of both sensor and actuator has usually to respond within the millisecond. The addition of security should provide a response performance at least similar to the one provided by current car IT system. Besides it should not increase the risk of error or of blocking the system.
2. *Resource limitations*: For cost-efficiency, the processing power and storage capacities of the ECU are assessed and limited to their minimum. The security should also be consistent with this fact and be optimized in order to use as little resources as possible.
3. *Start-up delay*: The ECU software needs to be ready within a short delay after the engine started. This requirement limits the possibility of initializing a security configuration at each start, e. g., key exchange, complex policy negotiation.
4. *Power saving*: Like the engine, the software should be energy-efficient. It is usually considered that 100W of continuous power correspond to 0,1l/100km of fuel consumption.
5. *External communication*: The number of standards for C2X communications is quickly increasing. The security should be adaptive to the used standards, their evolution and provide an optimal protection of the car.
6. *Software development*: The software of a car is increasingly complex. Its development involves not only the car manufacturer but also numerous subcontractors

and has to be adapted to each ECU platform. Besides its development is generally integrated within an iterative process, which requires multiple phases of testing and modifications. As a consequence, the development should be modular and allow the security part to be under the responsibility of a security expert team. More information about automotive software development is provided later in Section 5.1.3.

7. *External infotainment applications*: The development of on-board TPAs, CE-based applications and other online services cannot be controlled by a car manufacturer. The on-board security architecture should not be dependent on these external applications to provide any security support. In addition, the API available for these cases should provide good usability, limit the system complexity and be as security-unaware as possible.
8. *Life cycle*: Cars have a long life cycle, i. e., around 15 years. The software should be maintained all along this time lapse and should provide efficient mechanisms to be updated, e. g., rekeying process, policy update, addition of new C2X security protocols.

2.5 Summary

Current automotive technologies showed their limit both in term of functionality and security. The evolution of the automotive E/E architecture towards a full Ethernet/IP network could on one side provide better performance and flexibility to achieve new use cases and on the other side could make use of mature security protocol from the Internet world. However based on the attacker model and threats defined in this chapter, such an evolution may not be sufficient. A secure software architecture as well as a formal access control model for on-board communications, TPAs and C2X communications has still to be defined.

Chapter 3

Automotive IP-based Security Architecture

As motivated in Chapter 2, Ethernet/IP and associated security protocols will be extensively used by car manufacturers as standard for on-board communications. However the software architecture and security for on-board and C2X communications have not been standardized yet.

This Chapter provides an overview of the software basis of a security framework, providing secure communications and a secure execution environment. The architecture of an automotive security middleware is described in Section 3.1. After which, specifications about the security communication proxy are given in Section 3.2. Finally Section 3.3 discusses the benefits of this security architecture and pinpoints its shortcomings.

As mentioned earlier in Chapter 1, part of this work on security middleware architecture occurred within the SEIS project. The author led the middleware security work stream and reports here part of his results in Section 3.1.2. The modular architecture distribution, the modules for policy and authentication management are his original contributions. The modules for secure channels, crypto-services, key management and intrusion detection are inspired from SEIS [25] and adapted for the need of this thesis.

Parts of this chapter were previously published in *Driving Middleware Towards a Secure IP-based Future* [24], *Automotive Proxy-based Security Architecture for CE Device Integration* [27], *Middleware-Based Security and Privacy for In-car Integration of Third-Party Applications* [26], and *Middleware-based Security for Hyperconnected Applications in Future In-Car Networks* [23].

3.1 Middleware Security

A main challenge when designing security for cars is to provide homogeneous and adaptive solutions performing on a large variety of embedded devices and operating systems. For this purpose, the application layer and especially the middleware are optimal work bases to facilitate the definition of a security layer common to every ECU. The middleware is a software component present on every ECU and managing the communications

between applications of a distributed environment. In addition to ease the ECU interactions, this software layer can leverage its distributed architecture and enforce security as well.

This section firstly provides a general definition of the middleware layer and then discusses the automotive specifications it should comply with in Subsection 3.1.1. Subsection 3.1.2 then presents the architecture of the Security Middleware Extension (SME) and provides a thorough description of each of its modules. Afterwards, Subsection 3.1.3 proposes the description of a functional use case and highlights the general functioning of the SME and its modules. This chapter only considers the performance factor at a qualitative level. A quantitative assessment is later provided in Chapter 5.

3.1.1 Automotive Middleware

The middleware abstracts the communication management from the application level. Usually the development of distributed applications is a very expensive, time-consuming and error-prone task when using network OS primitives. As a consequence, the separation of the application logic from the communication management greatly facilitates the mission of the developers. A team of middleware experts focuses on developing the software basis for communication and security, which provides optimized interfaces to couple all necessary automotive applications. At the software level, the application acts as if all remote functions and resources were available locally. By following the formal definition given by Issarny et al [96], the middleware should define:

- an *Interface Definition Language (IDL)*, i. e., a programming-language-independent syntax and semantic allowing to describe the communication interfaces;
- a *high level addressing scheme*, i. e., a convention of the network to locate the resource, in the considered case it will be IP addresses and machine ports;
- a *coordination model based on an interaction and paradigm semantics*, i. e., serialization rules, which allow remote applications to produce and consume streams of data. Practically these rules allow a complex object on a first platform and in a specific programming language to be transformed into an ordered series of data blocks that can be deserialized on a second platform using its own programming language;
- a *transport protocol*, i. e., a protocol to achieve the communications between two remote platforms, in this case UDP/IP or TCP/IP;
- a *naming and discovery protocol*, i. e., a protocol including conventions to publish and discover resources, e. g., via broadcast communication or via a central server and multicast, like the Bonjour protocol [6].

The architecture and requirements for automotive middleware have already been specified in Chapter 2 and in [178, 179]. This section only mentions here some of their main

characteristics. In addition to an efficient and flexible serialization, the automotive middleware will provide RPC functionalities for bidirectional communications as well as a Publish/Subscribe functionality allowing the mapping of functions similar to the ones provided by the MOST notifications and the CAN signals. UDP and TCP will be used as standard transport protocols depending on the application requirements.

Figure 3.1 presents the different components of an on-board middleware in the ISO/OSI model. The middleware concerns the representation (L5) and session (L6) layers, on top of the TCP/IP stack. The *session semantic* layer belongs to the session layer as well and specifies the transmission type (i. e., synchronous or asynchronous) and the function call error semantic (e. g., maybe, at-least-once, exactly-once). The *serialization* layer, in the representation layer, implements the middleware coordination model. Practically, the middleware comprises a skeleton code written in a binding language (e. g., C or Java), which implements the serialization/deserialization processes as specified by the communication interface definitions. The middleware development can take advantage of custom compilers automatically generating the skeleton code based on an IDL-based definition of the communications interfaces. For an overall communication security, the security framework should be able to link its enforcement to all layers of the communication stack involved by the chosen security protocols.

All ECUs of a same car run the same middleware, i. e., share the same serialization rules. However different platform specifications in terms of resource and performance (e. g., a sensor versus a HU) may allow the more powerful ECUs to take part in more complex use cases than the weaker ones. Weckemann et al [178] defined three interoperable middleware categories allowing different levels of complexity and communication paradigms: Maximum, Medium and Minimum. In order to be consistent with this previous work, the approach presented here follows the same methodology and establishes a three-level security solution described in Section 3.3.

Related work about middleware security: After a thorough evaluation phase at a pure functional level [177], car manufacturers came to the conclusion that they had to build up their own middleware. Automotive middleware should be modular, flexible and interoperable. The same requirements apply to security. Obviously traditional middleware architectures already provide multiple security mechanisms [3]. But depending on the use cases and functional requirements they are subject to, they may offer very different security levels: from complex authorization models and strong communication security for poor performance and high security [16] to simple static ACL and very simple authentication schemes for high efficiency but low security [138]. The automotive context is extremely demanding and its real-time requirements make most middleware unsuitable [183]. Besides they usually do not provide the necessary security modularization or interoperability required to establish distinct levels of middleware security.

Since a few years, the avionics industry is confronted to similar security issues [167]. Like cars, they wish to open up their connectivity panel in order to intensify com-

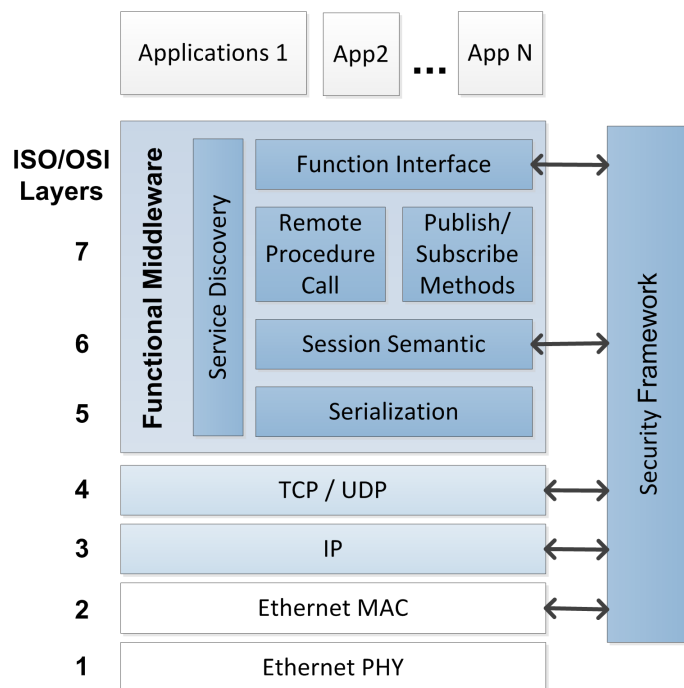


Figure 3.1: Overview of an automotive IP-based middleware and its integration within the ISO/OSI model. In this figure MAC means media access control.

munications between the plane and the outside (e.g., ground-based or plane-to-plane communications) and directly tether infotainment services of customers' devices to their on-board network. Despite few available research works, the approaches are not focusing on the software or the middleware level, but rather tend towards firewalls, physical network separations [155] and the use of strong security protocols [168].

3.1.2 Security Middleware Extension (SME)

In comparison to the CE industry, car manufacturers may seem to be conservative even sometimes reluctant to technical innovation, but the complexity of the car architecture and its functional requirements make the integration of new features very challenging. Each car manufacturer may develop and optimize one middleware for a specific car model, even sometimes depending on the provided infotainment or ADAS features. Therefore instead of developing a security solution for a specific middleware, car manufacturers of the SEIS project decided to cooperate and design a generic security framework, developed as an extension that can be easily coupled to most automotive IP-based middlewares [24].

The SME provides the bases for all necessary security services, i.e., enforcement of a secure application runtime and establishment of secure communication channels. The SME is composed of several modules. A module is an abstract representation of a spe-

cific security service, independent from its implementation characteristics, i.e., whether the whole module is implemented on one ECU or distributed over several. The SME communicates with the functional middleware via the Security Abstraction Layer (SAL). The concrete implementations of the security mechanisms and protocols are provided by the plug-ins, encapsulated within the modules. The SME architecture is designed to offer scalability, flexibility and middleware independence and as a consequence follows the next principles:

- *Suitable modularization*: each module is configured based on specific requirements, e. g., use cases, security levels, QoS, expected error rate, and therefore can be optimized for its target purpose. A suitable modularization allows to define different versions of each module providing an adapted choice of security plug-ins.
- *Abstraction of the security interfaces*: the assembly of complex and modular systems requires the definition of suitable and abstract interfaces. They simplify the module interconnections, their verification and clarify the coupling of security mechanisms to their enforcement locations, i. e., at the application-, middleware-, transport-, network- or physical-levels.
- *Configurability*: the car and its security infrastructure should allow static and dynamic configuration updates during the car assembly or later during its daily usage.

Designing an application with security in mind is a very complex task. The SME aims at simplifying and automating this process. Each application and communication is mapped with a set of both functional and security requirements, that can be for example declared during the communication interface description and the automatic generation of a skeleton code or at the application level, directly during the function call. The SME allows car manufacturers to mitigate the risk of security misconfiguration in the application.

The SME consists of six modules specialized in six different security purposes. Figure 3.2 presents the communication interactions between modules and with the functional middleware. The modularization is based on a three-layer organization:

- The *Decision Layer* provides security decisions by means of security policies and detects any security violation. It consists of the Policy Management Module (PMM) and the Intrusion Detection Module (IDM).
- The *Communication Layer* is in charge of the establishment and verification of the secure communication channels. It manages all plug-ins necessary for the protocol implementations and related filters. This layer consists of the Secure Channel Module (SCM) and the Authentication Management Module (AMM).
- The *Cryptographic Layer* is responsible for the key management and the cryptographic processing management. It consists of the Crypto-Service Module (CSM) and the Key Management Module (KMM). Depending on the expected security

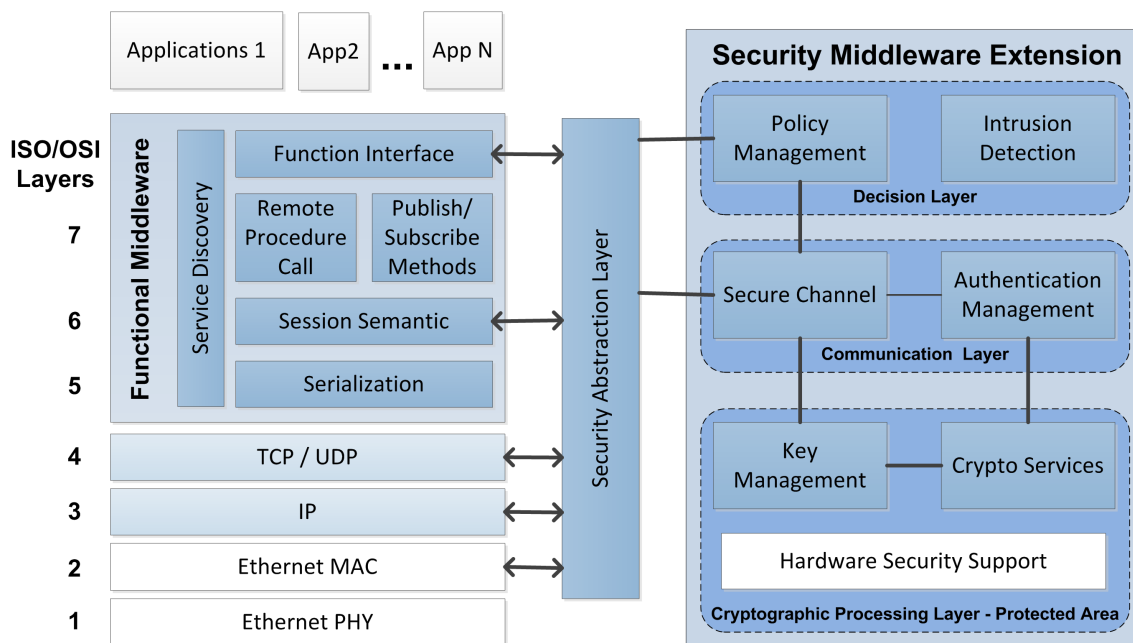


Figure 3.2: Connections between functional middleware and SME. In this figure MAC means media access control.

level, the security implementation of this layer may be located in a HSM. In this case only the API managing these functionalities are in the software and as a consequence part of the SME.

Such a modularization allows to optimally leverage all software and hardware capacities of an ECU and to establish different middleware security levels. A detailed description of each module and levels are provided in the rest of this chapter. Extensions of the API presented here are provided in Bouard et al [25].

3.1.2.1 Crypto-Service Module (CSM):

Most security mechanisms rely on a couple of security primitives and algorithms, which have to be carefully chosen depending on the layer of enforcement and the asset they aim to secure. These primitives and algorithms are based on the usage of sensitive and secret cryptographic material, i. e., the keys. In most academic work, the integrity, the authenticity and the confidentiality of such a material is a basic assumption to secure a system, e. g., a car. However, this is usually one of the most challenging assets to protect: security mechanisms are as secure as their base primitives, their implementation, the hardware and the software they are executed on.

Based on various types of security primitives and algorithms, the CSM provides all cryptographic services for encryption/decryption, digital signature generation/verification and processing of MACs. Processes invoking the CSM can parameterize their re-

quests and specify which algorithm, primitives to use and which data to process. Depending on the ECU security properties, e. g., secure key storage, protected processing environment, and its computation resources, the cryptographic services can be implemented locally on the ECU or centrally on a dedicated *Security Master ECU*. For the latter case, the request to the local CSM is forwarded to the *Security Master ECU* via a secure network channel. The CSM abstracts the cryptographic processing from its actual implementation and location.

Regrouping the security processing in a single service offers several advantages. Firstly, this service is the only one requiring a direct access to the secret cryptographic material. All other processes, even the ones ruling as the owner of the key, do not get access to it and only need a key handle, i. e., an identifier, to express which key to use. Exchanges of key handles and additional parameters replace exchanges of keys in plaintext and as a result reduce the need of confidentiality and the risk of disclosure. The actual keys are only communicated from the KMM to the CSM and use a dedicated protected channel. Ideally such level of security can be achieved when integrating CSM and KMM in a protected environment, e. g., in a HSM [49] or in a Secure Hardware Extension (SHE) [61, 184].

Secondly, the pooling of all cryptographic function and material is beneficial as well. It allows an easier verification and enforcement of secure programming guidelines, e. g., for protection against side-channel attacks. The runtime security management is also simplified; it allows to directly identify the right implementation on the right platform. Maintenance and addition of new algorithms and primitives can be administrated centrally for a better crypto-agility. Cryptographic processing should always be performed in a secure and potentially isolated environment. Secure CSM approaches may leverage secure virtualization with hypervisor and strong separation of memory between privileged and unprivileged processes. If an additional level of security is required, HSM solutions may also be used and involve access restricted processing units reserved for the cryptographic processing only [181, 31]. For real-time environment with resource-intensive primitives and algorithms like asymmetric cryptography with large keys (e. g., RSA [151], ECC [101, 123]), specific HSMs may provide hardware acceleration performing quicker than software modules. As a consequence, the CSM, responsible of all cryptographic services, is a crucial component of the trust anchoring of the whole car. A part of the CSM API is described in Table 3.1.

3.1.2.2 Key Management Module (KMM):

The KMM provides a secure access to the cryptographic primitives, i. e., encryption keys and other security credentials necessary to all cryptographic processings. The KMM is tightly coupled to the CSM, together they form a trusted zone, where the actual primitives can be securely exchanged. The security primitives cannot leave this zone. If an application or a security module other than the CSM requests a key, the KMM answers indirectly and provides the related key handle. The key handles can be used to

Table 3.1: Partial API of the CSM (Exceptions are omitted for a better clarity).

• generateAuthData	
byte[]	generateAuthData(int mode, int handle, byte[] data)
Description	returns the MAC or the digital signature of the argument <code>data</code>
Parameters	- <code>mode</code> : the type of signature or MAC to generate - <code>handle</code> : the key/credentials to use - <code>data</code> : the data, from which the result is processed
• verifyAuthData	
boolean	verifyAuthData(int mode, int handle, byte[] data)
Description	returns true if the signature or MAC is valid, false otherwise
Parameters	- <code>mode</code> : the type of signature or MAC to verify - <code>handle</code> : the key/credentials to use - <code>data</code> : the MAC or signature to verify
• payloadEncrypt	
byte[]	payloadEncrypt(int algo, int handle, byte[] data)
Description	returns an encrypted payload
Parameters	- <code>algo</code> : the encryption algorithm to use - <code>handle</code> : the encryption key to use - <code>data</code> : the data to encrypt
• payloadDecrypt	
byte[]	payloadDecrypt(int algo, int handle, byte[] data)
Description	returns a decrypted payload
Parameters	- <code>algo</code> : the decryption algorithm to use - <code>handle</code> : the decryption key to use - <code>data</code> : the data to decrypt

invoke cryptographic functions of the CSM. The CSM can trigger the `getKey()` function by providing a key handle and retrieve the key it requires to complete its cryptographic action.

One of the main tasks of the KMM is to hide the cryptographic material from any unauthorized party. Keys and other credentials are sensitive information that should be protected accordingly, however this protection also depends on the ECU capabilities. An ECU, physically equipped with a HSM, should store its sensitive material in the encrypted storage of the HSM and keep the encryption key secure, i. e., thanks to physical protection providing tamper evidence (e. g., physical seal), resistance (e. g., shielding), detection (e. g., sensor) or response (e. g., key erasing). If no HSM is provided, the KMM should implement additional measures to protect its keys, like hiding them in the ECU flash memory. Obviously this last approach is less secure and only increases the attacker's effort to find and retrieve the keys.

The automotive world is extremely cost-driven, the number of HSM in a car is therefore expected to be kept at its minimum. Thus an ECU, equipped with a HSM should be able to offer specialized security services to others which are missing them. One of these services is to provide a remote secure storage for cryptographic keys. These "key servers", a.k.a. *Security Master ECUs*, can store cryptographic material that is too sensitive to be stored in an unsecured flash memory. Interactions with the "key servers" can be integrated into the local KMM of a non-HSM-equipped ECU, in order to hide the complexity of remotely requesting a key access. A part of the KMM API is described in Table 3.2.

3.1.2.3 Secure Channel Module (SCM):

In current and future connected cars, new features generally result from the interconnection of functions present across several ECUs. Communication interfaces are defined during the car development phase in order to facilitate the exchanges of information and the invocation of remote functions. On-board communications are generally use-case-specific and have precise requirements for QoS but also for security. The middleware approach provides clear and simple APIs, which allow to transmit complex objects expressed in a certain programming language and to also specify certain requirements. As said earlier, most of these approaches lack the features to express security in a flexible manner. Instead, the security tasks are generally left to the application developer, which have to implement their own custom solutions directly at the application layer. Designing and implementing security is not trivial and should therefore be part of the middleware as well.

The SCM aims at solving such an issue by offering to the application a transparent communication service which meets its security requirements. The SCM hides the complexity of the security mechanisms by offering a simple API, e. g., close from the standard POSIX socket API [82]. In order to establish a new communication channel, the application invokes the function `openSecConnection()`, which specifies the target

Table 3.2: Partial API of the KMM (Exceptions are omitted for a better clarity).

• getKey	
byte []	getKey(int handle)
Description	returns an encryption key or some security credentials to a CSM, but only a key handle to other entities
Parameter	- handle : the identifier of the key to retrieve
• createKey	
int	createKey(int storage, int type, int size)
Description	creates a new key, stores it and returns its handle
Parameters	- storage : the identifier of the storage to use (HSM protected or not) - type : the type of the key to create (symmetric vs asymmetric) - size : the size of the key to create
• storeKey	
int	storeKey(int storage, byte[] key)
Description	stores a given key and returns its handle
Parameters	- storage : identifier of the storage to use (HSM protected or not) - key : the encryption key to store

application and passes the desired security parameters as argument. These requirement arguments clarify the security properties of the communication to establish. They may define whether the communication should be authentic, integrity-protected or confidential and which security credentials to use. Based on these requirements, the SCM determines the most suitable communication protocols to use. The specification of such arguments is optional, if nothing is provided the SCM may request the “by default” communication properties from the PMM.

The SCM coordinates the action of the SME modules. It conducts the AMM during the authentication process of the communication establishment. It provides the CSM and KMM with cryptographic tasks to perform. Finally it cooperates with the PMM, in order to comply with the ECU security requirements and policies. A part of the SCM API is described in Table 3.3.

3.1.2.4 Authentication Management Module (AMM):

The AMM takes part of the authentication process of every internal or external communicating entity. Its tasks are twofold: accounting and enforcing. Implemented locally on every ECU of the car, it stands in the middle of the communications between the SCM and the CSM and supports them during the authentication process.

Regarding its accounting role, the AMM establishes a mapping of the authentication status of every communicating entity: the authentication scheme, the status of the au-

Table 3.3: Partial API of the SCM (Exceptions are omitted for a better clarity).

• openSecConnection

int	<code>openSecConnection(int ECU_ID, int auth_level, int conf_level, int int_level)</code>
Description	opens a secure channel to ECU_ID and returns a channel identifier
Parameters	<ul style="list-style-type: none"> - ECU_ID: Identifier of the ECU to communicate with - auth_level: required strength of the authentication scheme - conf_level: required strength of the encryption - int_level: required strength of the integrity mechanisms

• secListen

void	<code>secListen(int interface, int auth_level, int conf_level, int int_level)</code>
Description	opens a listening interface
Parameters	<ul style="list-style-type: none"> - ECU_ID: Identifier of the ECU to communicate with - auth_level: minimal strength of accepted authentication schemes - conf_level: minimal strength of accepted encryption types - int_level: minimal strength of accepted integrity mechanisms

• forward_in

boolean	<code>forward_in(byte[] data)</code>
	forwards the data received from a remote peer to the functional middleware, returns true if the middleware of the application received it, false otherwise
Parameter	- data: data to pass to the application

thentication protocol completion and the security credentials that were used, i. e., the handles. Its role includes also the management of the authentication security material, e. g., requesting new cryptographic material, revoking security authentication tokens or certificates. On a HSM-equipped ECU, the AMM administrates the accounting features related to hardware-based remote attestation. In addition, the AMM supports the IDM by sending it regular reports about the system authentication status for intrusion detection analysis. When located on the edge of the on-board network and taking part of C2X communications, the AMM is also in charge of the pseudonym management.

For every security protocol, the authentication mechanisms play an essential role and give a proof of origin to every exchanged frame. As an enforcer, the AMM task is about synchronizing the KMM and CSM, so that they can generate and verify the appropriate authentication fields at the right moment. A part of the AMM API is described in Table 3.4.

3.1.2.5 Policy Management Module (PMM):

The PMM is responsible for the policy management and for every policy decision process of the SME, middleware and application layers. Its role is mostly consultative; the application, middleware or security module requesting a policy decision has to enforce the decision itself. With support of the IDM, the PMM can make sure that no on-board entity is misbehaving and can actively react to stop or limit the impact of a security breach, e. g., with process isolation or recovery process. The PMM is a module implemented on every ECU. This module should locally dispose of all necessary policies and context information in order to reduce the latency and the risk of error. Like the KMM for the keys, the PMM abstracts the policy management and the remote fetching of policies present on other ECUs. In order to limit the system complexity an redundancy, a central interface is responsible to deploy the policies and security configurations during a security update. These security updates are periodically performed all along the car lifespan, when the car is stopped.

Designing access control models (i. e., policy management and policy format) for complex and real-time systems can be challenging. Suitable tradeoffs between decision-making performance and level of expressiveness have to be defined. With respect to the car functional requirements for low latency and error rate, the system enforces two policy formats:

- **Middleware-level policy.** These policies are requested and enforced by the middleware. They defined which communications are authorized, over which protocols and sometimes with which security credentials. They are static and quite simple. A more sophisticated version of them is provided in Chapter 4.
- **Application-level policy.** These policies are requested and enforced by the application, i. e., directly given to the application through the function interface `getPolicyDecision()`. They are more expressive and need application know-

Table 3.4: Partial API of the AMM (Exceptions are omitted for a better clarity).

• startAuthentication_Server	
int	startAuthentication_Server(int ECU_ID, int auth_method, byte[] data)
Description	verifies the received authentication fields and returns the authentication status of ECU_ID (authenticated, non-authenticated, pending or blacklisted)
Parameters	- ECU_ID: Identifier of the ECU to communicate with - auth_method: authentication method/schemes to verify - data: MAC or signature that has to be verified
• startAuthentication_Client	
byte[]	startAuthentication_Client(int ECU_ID, int auth_method)
Description	returns the suitable authentication data to start the authentication and logs the authentication status
Parameters	- ECU_ID: Identifier of the ECU to communicate with - auth_method: authentication method/schemes to verify
• finishAuthentication_Server	
byte[]	finishAuthentication_Server(int ECU_ID, int auth_method)
Description	returns the suitable authentication data to finish the authentication and logs the authentication status
Parameters	- ECU_ID: Identifier of the ECU to communicate with - auth_method: authentication method/schemes to generate
• finishAuthentication_Client	
int	finishAuthentication_Client(int ECU_ID, int auth_method, byte[] data)
Description	verifies the received authentication fields and returns the authentication status of ECU_ID (authenticated, non-authenticated, pending or blacklisted)
Parameters	- ECU_ID: Identifier of the ECU to communicate with - auth_method: authentication method/schemes to verify - data: MAC or signature that has to be verified

Table 3.5: Partial API of the PMM (Exceptions are omitted for a better clarity).

• getPolicyDecision

int	<code>getPolicyDecision(int ID, int action, byte[] action_arg)</code>
Description	returns the decision of a policy evaluation (0 granted, 1 refused, 2 no available answer, 3 invalid)
Parameters	- ID : Identifier of the ECU/application requesting an authorization - action : Type of the requested resource (e. g., communication, file access) - action_arg : appropriate complement of the requested resource (e. g., protocol, resource identifier)

• getPolicy

byte []	<code>getPolicy(int policy_ID)</code>
Description	returns the requested policy(ies)
Parameter	- policy_ID : Identifier of a policy or a group of policies

ledge. They generally regulate access to a resource, e. g., database, file, physical mechanisms. They are particularly adapted to non-time-critical applications, e. g., with user interaction or dynamic tethering of external devices like smartphones or online services.

This work does not recommend any policy language. Expressive policy languages like XACML [142] often necessitate slow evaluation engines and as a consequence need to be evaluated on a case-by-case basis. An example of light automotive policy language based on the ASN.1 format can be found in Idrees et al [81]. A part of the PMM API is described in Table 3.5.

3.1.2.6 Intrusion Detection Module (IDM):

The IDM is about protecting the car against external tampering and attacks targeting a single or multiple ECUs. This module is distributed over several ECUs. Each implementation does not run necessarily the same intrusion detection mechanisms, but they cooperate with each others. The cooperation between all on-board IDMs establishes a distributed IDS embedded at the middleware level. All collected raw data can be transmitted to a central IDS nodes, a.k.a. *Security Master ECUs*, performing their analysis and taking further decisions about which countermeasures to launch. An example of automotive countermeasure is to write an entry in the log-file. But it could be a more active process, e. g., sending warning signals to the driver and notify her about an on-going attack [74].

Like in the traditional computer world, the IDM can include multiple common mechanisms allowing it to detect an intrusion [10]:

- **Host-based IDS:** These mechanisms mostly monitor the internal vitals of a computer and log all unexpected and unauthorized behaviors for later analysis. Such an IDM can be customized to the automotive needs of an ECU, e.g., for an intensive processing of multimedia data stream or for processing of safety-critical sensor data. The car architecture is quite static, the IDM can therefore be aware of the repartition of computational resources allocated for each purpose (e.g., infotainment, ADAS, power-train, cabin control) and detect any anomaly. A host-based IDM can in addition participate in the cooperative IDS by periodically requesting remote attestations from other platforms [103].
- **Network-based IDS:** Such an approach monitors the network traffic. This relies on signature-based mechanisms, comparing the traffic with pre-configured and pre-determined attack patterns. As mentioned in the precedent bullet, the staticity of the ECU can be applied to the network exchanges as well. The IDM can also set boundary rates to certain types of exchanges and detect an abnormal traffic increase or decrease [75]. When implemented on the C2X gateway, the IDM should be able to monitor external communication interfaces and detect remote DoS attacks as well.
- **Introspection-based IDS:** This approach allows the IDS to monitor a system from an external point of view and is particularly used for virtualized environment [62]. The IDS, implemented on the physical ECU, gathers information provided by the hypervisor and then performs recognition tests of machine instruction patterns in order to assume the state of the virtual partition. An approach for introspection-based monitoring is later investigated in more details for TPA monitoring in Chapter 4.

In addition to these three locally enforced approaches, a central IDM can leverage the other security modules and collect their logs. The AMM could inform the IDM about unsuccessful authentication attempts with invalid or revoked security credentials. The PMM could report every policy infringing situation it has been asked to evaluate. Finally the SCM could notify all unauthorized attempts of communication establishment.

Considering the automotive cost and performance requirements, it seems clear that only ECUs handling sensitive and non-time-critical assets will be equipped with sophisticated IDM, i. e., host- and introspection-based. The addition of network-based IDS for all on-board and C2X gateway or router ECUs is also strongly recommended. A part of the IDM API is described in Table 3.6.

3.1.2.7 Security Abstraction Layer (SAL):

The SAL is optional and not necessary if the middleware can directly make use of the SME implementation. The SAL provides a logical interface linking applications and middleware to the SME in specific cases, e.g., when middleware and SME are not written in the same language or not compiled together. A generic API (e.g., Inter-Process

Table 3.6: Partial API of the IDM (Exceptions are omitted for a better clarity).

• notifyViolation	
boolean	<code>notifyViolation(int module_ID, int ids_method, int event_type, byte[] event_arg)</code>
Description	returns true if the event is considered as an intrusion and launches, adapted countermeasures, or returns false otherwise
Parameters	<ul style="list-style-type: none"> - <code>module_ID</code>: Identifier of the emitting IDM module - <code>ids_method</code>: Method of intrusion detection (network, introspection) - <code>event_type</code>: type of the event (unauthorized access, communication) - <code>event_arg</code>: additional information about the event to notify
• sendAnalysisData	
boolean	<code>sendAnalysisData(int module_ID, int data_type, byte[] data)</code>
Description	analyzes raw event data from non-IDM and returns true if the data are accepted and treated, or returns false otherwise
Parameters	<ul style="list-style-type: none"> - <code>module_ID</code>: Identifier of the module providing the data - <code>data_type</code>: type of the data to analyze (log, network input) - <code>data</code>: raw data to analyze

Communications (IPCs) with the SME, or a JNI-based API [141] for communication between Java and C implementations) allow the developer to transparently contact the SME and take care of the module dependencies.

3.1.3 Functional Use Case and SME Management

Section 3.1.2 described in details every module of the SME and provided part of their API. This section presents a simple example of communications between two on-board applications and demonstrates the interactions between the functional application/middleware and the SME. This example considers the connection establishment process and data exchange between an on-board client and an on-board server. This use case features a bidirectional communication channel. The server is assumed to be already configured in order to receive incoming communications, i. e., the cryptographic material is in place and the serversocket is listening. On the other side, the client is assumed to be set up with valid cryptographic material allowing it to initiate a communication with the server.

3.1.3.1 SME Management – Client Side

Figure 3.3 presents a graphical sequence chart highlighting the runtime of the client. Only the local implementation of the modules is represented here. For a better understanding, cases where a module should perform a remote request for a key or a policy are

not considered. The runtime is pictured in four steps. Steps 1 to 3 describe the secure channel establishment, which starts with invocation of the `opensecConnection()` by the *Client Application*. Step 4 is launched by the function `send()` and describes the emission of data from the client to the server. For more simplicity the IDM, which is involved only if a security violation is noticed, is not drawn here.

Step 1 – Authorization: The `openSecConnection()` call is performed on the SCM. The SCM seeks for a policy decision to the PMM, which may authorize a communication with the requested server and the requested security configuration. If no security argument is provided by the *Client Application*, a “by default” configuration of the communication in addition to the policy decision is requested.

Step 2 – Authentication (1st phase): Once the SCM is sure that the connection is permitted, the second step, i. e., the authentication, may start. The SCM contacts the AMM, which launches the handshake process. The AMM calls the CSM with the appropriate key handle and a precise description of which type of MAC or signature to generate. The CSM then retrieves the necessary cryptographic material from the KMM, completes its tasks and sends the result to the AMM, which forwards it the SCM. The SCM plug-in, responsible of the chosen security protocol, performs the serialization of the *Session-Establishment Message*. The SCM passes it to the CSM for encryption and then sends the encrypted version to the server via the network.

Step 3 – Authentication (2nd phase): After reception of the *Session-Establishment Response* of the server, the SCM passes the packet to the AMM. The packet is then decrypted and the authentication fields are verified by the CSM. After what, the CSM provides the AMM and the SCM with its answer. The AMM notifies the successful authentication process in its log file. The SCM can also notify the *Client Application* about the success of the communication establishment.

Step 4 – Secure communication: The *Client Application* and SME went through the authentication process and session establishment and can now directly communicate with the server. The SCM sends directly the packets provided by the `send()` function call of the *Client Application* after having it encrypted by the CSM.

3.1.3.2 SME Management – Server Side

In a similar manner, Figure 3.4 presents the same functional use case, but this time from the server point of view. The *Server Application* gets initiated through a routine like `seclisten()`. The SCM waits for an incoming connection at least as secure as the configuration provided by the function arguments.

Step 1 – Authorization: Like in the previous case, the session establishment process starts, with the SCM asking the PMM whether the communication channel with the security configuration proposed by the client side may be opened.

Step 2 – Authentication (1st phase): After a positive acknowledgement, the SCM is allowed to decrypt the packet via the CSM, which automatically loads the appropriate key from the KMM. Via the AMM, the SCM has the authentication fields verified by

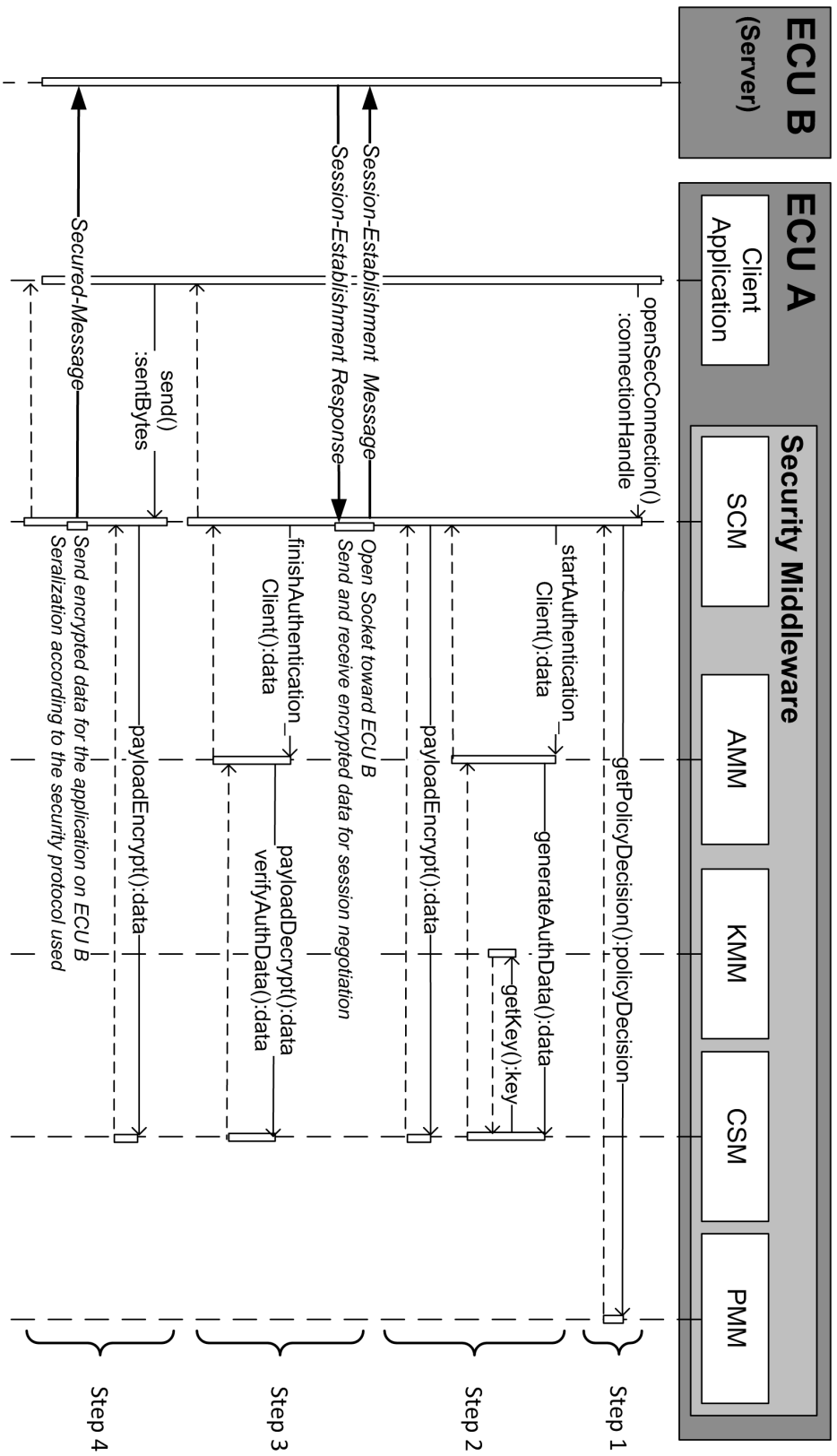


Figure 3.3: Functional use case: Open connection & data sending (client side).

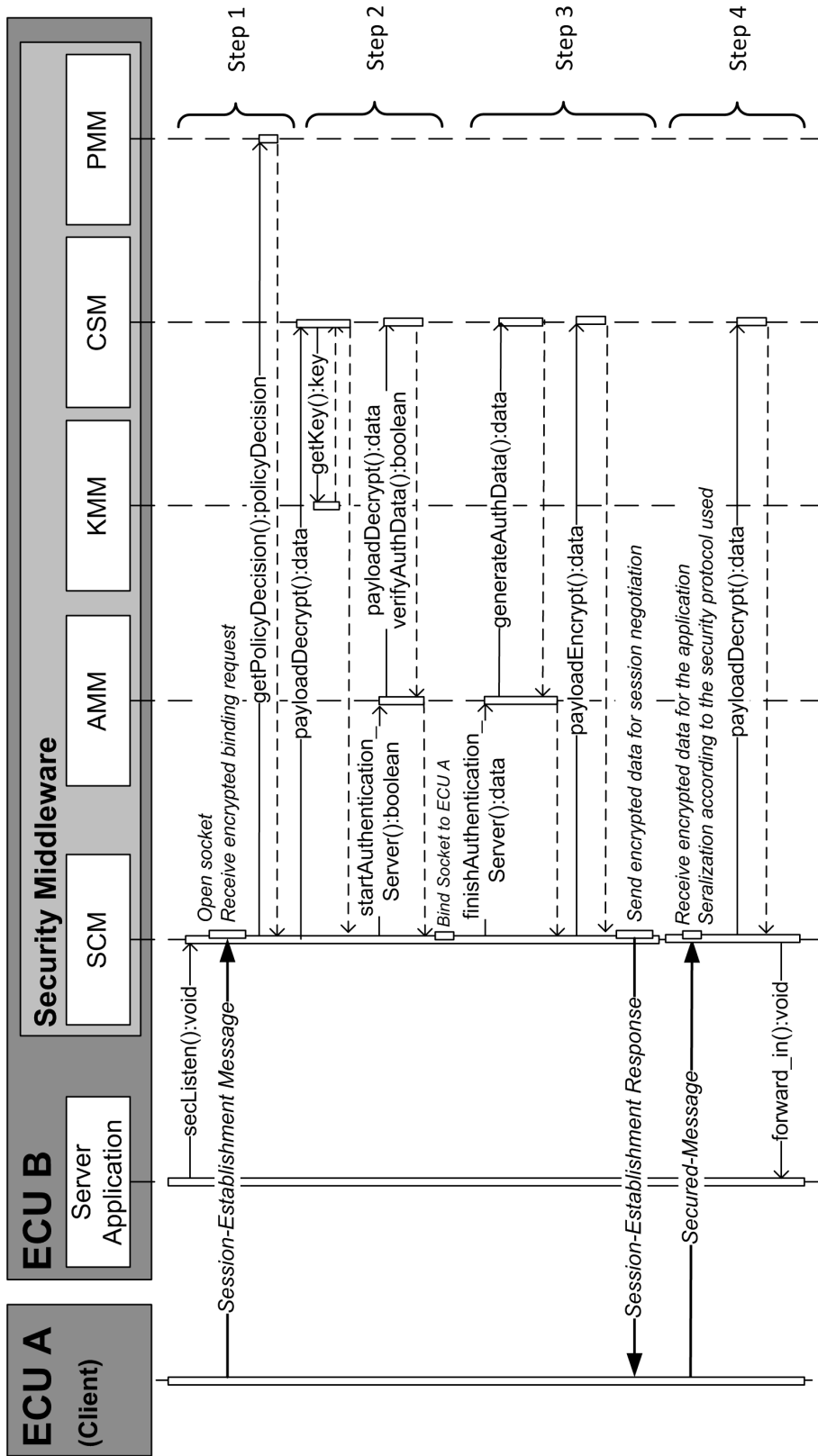


Figure 3.4: Functional use case: Open connection & data sending (server side).

the CSM. If these fields are valid, the SCM allows the socket to be bound to the client and to receive data from it.

Step 3 – Authentication (2nd phase): The SCM builds the final part of the handshake, i. e., the *Session-Establishment Response*, and asks the AMM to generate the appropriate authentication fields via the CSM. The SCM then serializes the packet, has it encrypted by the CSM and finally sends it to the client side.

Step 4 – Secure Communication: After completion of the session establishment process, the *Server Application* is ready to receive messages from the client. After the reception of a *Secured-Message*, the SCM can decrypt the message via the CSM and if it is valid, forwards it to the functional middleware of the *Server Application* via the function call `forward_in()`.

Additional comments: This example presents a very simple session establishment with a two-way handshake. More complex handshakes and key establishment protocols can be supported in a similar way. The SCM processes the additional message exchanges as in Steps 2 and 3 of the client and server. However, the automotive system is subject to drastic safety and performance requirements and generally cannot afford the latency and error risks induced by complex security mechanisms. The session establishments during runtime should be therefore kept simple and efficient. More complex features and dynamic key exchange should be mostly performed along the assembly line or during periodic system updates, when the car is stopped. As a consequence, the SME configuration of most platforms, especially the resource-limited ones, should be static and should encompass simple middleware-based policies, the setup of security associations for on-board IPsec channels between ECUs and the distribution of preshared keys.

3.2 Security Communication Proxy

Car manufacturers intend to centralize most C2X communication interfaces around a single Multi-Platform Antenna (MPA) [120]. More than just a super antenna, the MPA is a complete ECU directly connected to the on-board Ethernet network and equipped with a fully functional IP-based middleware. The MPA provides a great opportunity to build efficient security solutions, applicable to all C2X communication types and easy to verify and maintain.

This section firstly motivates why the MPA necessitates a special security design in Subsection 3.2.1, after what Subsection 3.2.2 introduces a first automotive approach for C2X Information Flow Control (IFC). Then, the MPA security architecture and in particular specificities of its SME are presented in Subsection 3.2.3. Finally, the concept of Security & Trust Level (STL), which provides a security taxonomy for external communication partners, is described in Subsection 3.2.4.

3.2.1 Towards Secure Automotive Proxy-Middleware

The on-board middleware may be optimized for static and resource-limited configurations, but limited as for dynamic C2X use cases involving the MPA. The integration of a large variety of external communication partners requires additional features for a more efficient service discovery and flexibility. In addition controlling information flows in distributed systems like cars is essential for holistic security solutions. ECUs internally exchange genuine packets and therefore only require secure communication channels and simple access control mechanisms. But the integration of untrusted communication partners, over which the car manufacturer has no control, is quite risky and necessitates more complex security mechanisms.

CE devices, Internet services, RSUs and others cars are heterogeneous systems, whose functional and security capabilities depend on several factors, e. g., their OS, hardware. Car manufacturers cannot ban certain types of smartphones, competitor car models or trendy online services from connecting to their cars, just because they do not fit all of their requirements. Instead of restricting the car access to specific classes of devices or services, the on-board architecture needs to be adaptive and so does the underlying security infrastructure. Cars and more specifically their MPA have to be able to integrate all external communicating entities over a wide range of media (e.g, Wi-Fi, LTE, 3G, IEEE 802.11p) and communication protocols (e. g., automotive- or web service-specific). At the same time, the complexity of the MPA should not exponentially rise. The car has to rely on the MPA and its programmable platform for managing the access between external and on-board networks in a secure way. For the rest of this work, the terms *security communication proxy* designate the software/hardware components of the MPA where the security is implemented.

More than just a packet forwarder, the proxy decouples every communication between inside and outside similarly to a NAT router. The decoupling mechanisms allow the car to use on the inside a limited set of optimal security protocols, while letting the choice of the outside protocol to third-party developers. Ethernet/IP will allow a bigger internal bandwidth able to carry large objects, will increase the number of remotely-available functions on every ECU and as a consequence will enable a large and demanding C2X traffic. However, the verification of both security requirements and packet validity for every message will not be possible only at the proxy level. The proxy cannot perform deep packet inspection and should stay unaware of the application level. But at the same time, the ECUs are never in direct contact with any external devices or services and are incapable of taking holistic security decisions. Therefore in addition to decoupling the C2X communication, the proxy should propose some cooperation mechanisms allowing it to support the ECU in its security decision process. This proxy architecture is suitable in this sense and allows to share the security enforcement between proxy and ECUs: while the proxy manages the external security protocols, the ECUs enforces their own security decisions based on additional context information provided by the proxy.

Related work about proxy security: Corporate network security and automotive on-board security present numerous similarities, especially when integrating mobile devices. Recently several companies took advantage of the Bring Your Own Device (BYOD) trend in order to lower their asset costs and employee efficiency, but also faced major security issues [126, 69], e. g., theft of industrial secrets, dysfunctions of their network infrastructure. Their approach mostly relies on strong authentication mechanisms and device integrity measurements in order to establish network connections or a VPN access [47, 40]. The integrity measurements usually check the version of the device spyware, antivirus and, if available, the presence of a secure element [47]. However, such approaches only regulate the internal network access and usually lack specifications for resources- and data-management.

Plane manufacturers want also to allow passenger to connect via their personal mobile devices to infotainment resources of the plane. But as mentioned in the previous section, plane security for the moment relies on a physical separation between critical and non critical networks [155]. In opposition to planes, automotive use cases often leverage simultaneous integration and interconnection of functions from critical and non critical domains and make such an approach not suitable.

3.2.2 Information Flow Control, a First Approach

Section 3.2.1 announced the notion of cooperation mechanisms between proxy and ECU. For this purpose, an in-band middleware protocol was extended and allows to exchange additional security metadata. Concretely the middleware header is extended with a new field specifying the security and trust context in which the data are exchanged between the car and an external communication partner. Instead of directly considering the privacy aspect of any single piece of information, the new middleware protocol focuses on the trust, a communication peer is granted by the car and intends to quantify it. Thus, the security aspect characterizes how secure the external communication is. On the other side, the trust aspect characterizes how trustworthy the remote device of online service is considered to be. This context in term of security and trust is called Security & Trust Level (STL).

In order to distinguish whether the STL describes the current communication situation or whether it describes a required situation in order to send a message to the outside, two types of STLs are defined:

- The **STL_{STATUS}**, generated by the proxy and enforced by the ECU, this STL allows the ECU to evaluate the risk it is taking if it processes the related message.
- The **STL_{REQ}**, generated by the ECU and enforced by the proxy, this STL allows the proxy to evaluate whether the message can be forwarded and whether it fits the requirements imposed by the middleware which produced the data.

The life cycle of the STL is graphically described in Figure 3.5. The STL taxonomy maps abstract security concepts and requirements to concrete protocols mechanisms. It

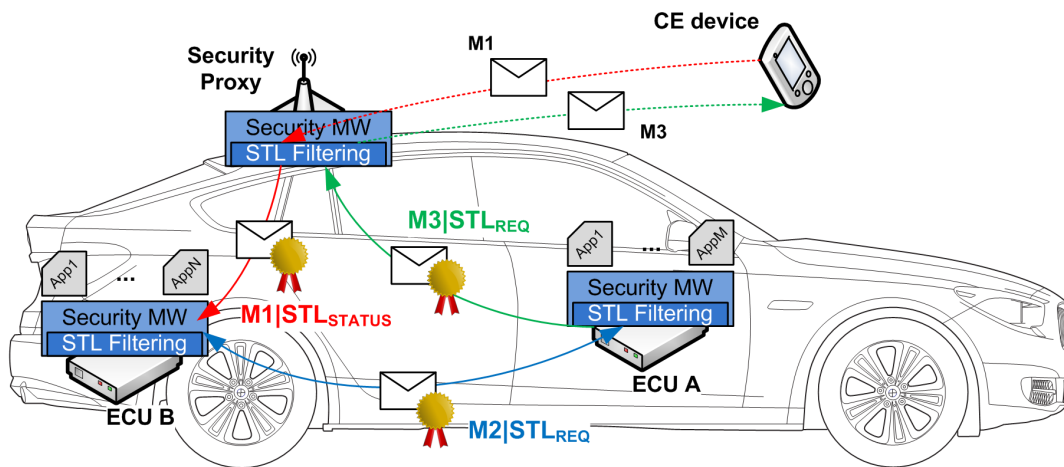


Figure 3.5: STL life cycle. Messages are symbolized by an envelope, the STL metadata by a medal. A STL_{STATUS} is only exchanged from the proxy to an ECU, whereas a STL_{REQ} can be exchanged from an ECU to the proxy or between ECUs. The figure features a CE device, but it can be also replaced by other external entities, e. g., online service, RSU.

allows an efficient and generic security enforcement at the ECU level, independently of the external protocol or situation specificities. More precisions about where the STL are enforced and how it is evaluated are given respectively in Subsections 3.2.3 and 3.2.4. Security for C2X communications relies here on the fact that the whole C2X traffic goes through the proxy. This assumption was already mentioned in the attacker model presented in Section 2.3.4. This point is however later discussed in Section 3.3.2.

3.2.3 Extending the SME for a Security Communication Proxy

This subsection describes (1) the architecture of the security proxy and (2) how the SME on an ECU can be STL-enabled for a secure proxy–ECU cooperation.

3.2.3.1 STL-enabled Security Communication Proxy

The proxy is implemented on the MPA and stands in the middle of every communication occurring between an internal entity and the outside. Unlike traditional NAT routers simply forwarding IP packets, the proxy really decouples the communications and may act like a translation interface between external protocols (e.g., HTTP) and internal middleware-based protocol.

The proxy is a critical element of the car, essential in all C2X communications and in direct contact with untrusted networks and external attackers. As a result, the proxy disposes of a strong security architecture equipped with a HSM for secure key storage and hardware-based cryptographic processing. Its SCM is designed to provide internal

communication interfaces over static IPsec channels and an external dynamic interfaces, compatible with a wide range of web-based and middleware-based protocols providing security or not. More than just handling the authentication process for both internal and external entities, its AMM also manages the car pseudonyms and their short-term certificates used during privacy-infringing use cases, e. g., for position broadcasting during an emergency braking. Regarding the C2X policy management, PMM and SCM of the proxy play a major role for the STL concept. For each external communicating entity, the PMM evaluates and transmits to the SCM a STL_{STATUS} to add to the middleware header of every inbound packet (M1 in Figure 3.5). The proxy is application unaware, but depending on the entity identity PMM and SCM can enforce a domain-based filtering on inbound messages, e. g., an online service for social network logged into the car, even on behalf of the driver, will not get access to any functions of the power train management. Inversely before forwarding an outbound message (M3 in Figure 3.5), the proxy makes sure that the STL_{REQ} provided by the ECU, at the origin of the message, is conform to the actual STL of the communication status, i. e., to the STL_{STATUS} . More details about STL evaluation and rules are provided in Section 3.2.4. Finally, as earlier mentioned in the section about the IDM, the proxy is equipped with a combination of IDS, e. g., host-, traffic- and introspection-based IDS, allowing it to be more resilient against external attackers.

3.2.3.2 STL-enabled On-board Security Middleware

Present on every ECU, the middleware relies on the SME for all security services. In order to enforce the STL policies, the SME architecture is only slightly modified. The main modifications concern the PMM for the evaluation of adapted STL-based policies and the SCM for the enforcement of the policy decision for both received and emitted packets.

Based on the received STL_{STATUS} , the PMM decides whether it is safe and authorized to process such a packet. The STL_{STATUS} informs the SME about the likelihood of an attack, e. g., an unauthorized data modification, while they were traveling over the external wireless network. Depending on the ECU capacity, the SME may chose a more protected execution environment in order to process the received data, e. g., parsing SQL-based access requests through a specific security parser or running JavaScript pieces of code in an isolated web browser.

On the other hand, a received STL_{REQ} , determines the sensitivity of the data contained in the message payload. The STL requires PMM and SCM to decide whether their application level is allowed to receive such data, i. e., whether their applications offer enough confidentiality guarantees to not leak the information. Inversely, when sending a message, the SCM extends the middleware header with a STL_{REQ} reflecting the sensitivity of the payload, i. e., industrial secret or private information of the driver. Generally data labeled with STL_{REQ} are produced by other ECUs and therefore do not present any integrity risk therefore only their confidentiality is considered.

For this chapter, all on-board communications are labeled with a STL_{STATUS} or a STL_{REQ} . As a consequence, even if an ECU or an on-board multicast address inadvertently forward a STL labeled message to the proxy, i. e., to the outside, the proxy can still enforce a suitable STL rule. The applications on top of the middleware are totally STL transparent, the STL enforcement happens at the SME level. Like most on-board policies, the STL-based policies are defined by car manufacturers at design time and are established based on the use cases the applications are involved in.

3.2.4 Security & Trust Level (STL) Taxonomy

Section 3.2 introduced the concept of STL and defined it as the security and trust context in which data are (STL_{STATUS}) or should be (STL_{REQ}) exchanged with the outside. The rest of this subsection proposes the evaluation of (1) the STL security aspect, (2) of the STL trust aspect and (3) the enforcement of STL rules.

3.2.4.1 Security Level (SL)

The Security Level (SL) is defined as a qualitative description of the security strength of an external communication. Concretely, a specific SL value is associated to each C2X security protocol. These protocols can be located on different layers of the communication stack and combined together, in this last case only the higher security level is taken into account. The different levels and the security requirements, they have to comply with, are characterized as follows:

$SL=0$ Communication protocols providing no or weak security or presenting exploitable design flaws. Here “weak” designates security mechanisms that can be broken in less than 4 hours, i. e., less than an average driving time, only attacks performed when the car is active/driven are considered.

Examples: Plaintext; WEP encryption; TLS+DES or RC4 with a 56-bits key;

$SL=1$ Communication protocols providing strong authentication of the external peers and data integrity, i. e., against unauthorized modifications.

Examples: WPA2 encryption; Message in plaintext protected by HMAC-SHA-1;

$SL=2$ Communication protocols as secure as $SL=1$ and in addition providing strong confidentiality, i. e., one secret key per user, no shared key between users.

Examples: TLS+AES; IPsec+AES;

$SL=3$ Communication protocols as secure as $SL=2$ and assuring the presence of a secure hardware element protecting the cryptographic materials of the external peer.

Examples: $SL2$ -protocol + remote attestation.

3.2.4.2 Trust Level (TL)

The Trust Level (TL) is defined as an abstract representation of how trustworthy an external data sender/receiver is. The notion of trust is usually defined as a mix between 3 components: reputation, reliability and security [161, 66, 116]. The security has already been considered, thus the TL focuses on the 2 remaining ones. The evaluation criteria of the TL should be clear and easy to assess. This work considers that data may only be misused, if they are (1) physically and (2) juridically accessible, i.e., (1) if the data leave the car and (2) if the receiver is legally allowed to endanger the user's privacy, e. g., data selling/forwarding, data stored on an unprotected server. The TL should reflect these risks and is evaluated based on the following criteria:

- **Criterion 1 (Cr. 1) “Local Usage”**: determines whether the data are limited to an on-board usage only.
- **Criterion 2 (Cr. 2) “Anonymization”**: determines whether data have to be anonymized, when released out, i.e., whether an external receiver may be able to trace back the identity of the car or of the user.
- **Criterion 3 (Cr. 3) “Jurisdiction”**: determines whether the external receiver is considered as a safe Place Of Jurisdiction (POJ), i.e., whether the servers hosting the online service are located in a country imposing a regulation protecting the user's privacy.

In order to determine the TL of an external peer, the simple binary decision tree presented in Figure 3.6 is used. Every criterion is evaluated iteratively, a “true” answer stops the process and sets the TL value. Highly sensitive data, like industrial secrets, are only reserved for a internal usage (Cr.1=true) and are labeled as requiring a very trustworthy usage (TL=3). Very sensitive data, like the car position, can leave the car but have to be untraceable (Cr.2=true), i.e., anonymized at the proxy level (TL=2). Data with a low sensitivity, like the driver's name, can be forwarded to services presenting a safe POJ (Cr.3=true, TL=1). While Cr.1 and Cr.2 are easy to assess and enforce by the proxy, Cr.3 needs to be specified by privacy experts, for example relying on literature inspecting the data protection laws of different countries [111].

The TL scale orders the level of trustworthiness, from data that can be sent to untrusted entity (TL=0) to data that have to be sent to on-board entity only (TL=3). Obviously such ordering is not exclusive, but it allows very trusted external entity to receive data that may be sent to untrusted peers, e. g., a peer able to receive data with a TL=2 is also authorized to receive data with TL=1 or 0.

In order to test the validity of this taxonomy, the TL of four realistic C2X scenarios are evaluated: the social network Facebook [58], which receives information from the car via the driver's account; Safebook [44], a privacy-aware peer-to-peer social network, which allows the user to locally store its data (i. e., in the car) and have full control about the release thereof; an online banking service having its servers in Germany, which allows

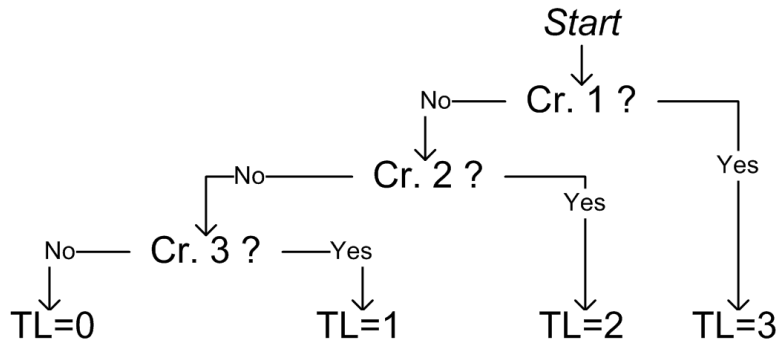


Figure 3.6: Binary decision tree for TL evaluation.

Scenarios	<i>Cr.1</i>	<i>Cr.2</i>	<i>Cr.3</i>	TL
Facebook	false	false	false	0
Safebook	false	false	false	0
Banking	false	false	true	1/0
LHW	false	true	-	2/1/0

Table 3.7: C2X Scenarios and assigned TLs. (LHW: Local Hazard Warning).

to manage from the car the driver’s account; and a Local Hazard Warning (LHW) car function, broadcasting safety messages, including the car position, to other road users. Results are provided in Table 3.7. Because of an unsafe POJ of its servers, i.e., the USA [111], Facebook should only receive non-sensitive data. The Safebook’s peers (i.e. “friends”) cannot be considered as being in a safe POJ and therefore are in the same case as Facebook. The Bank servers in Germany, a safe POJ [111], can receive TL=1- and TL=0-labeled data. As for the LHW scenario, other cars will receive the TL=2 labeled data only if the proxy is sure that they have been anonymized.

The TL taxonomy is relatively simple and seems to provide an efficient way to control the information release to the outside world. But further tests with more use cases should be performed. The TL criteria are very coarse, but give to car manufacturers an easy way to configure a “by default” privacy/trust-aware behavior. For a more flexible usage, users should be able to change the assigned TL of an online service, like a social network of their choice and allow it to receive some data with TL=1 as well.

3.2.4.3 The STL Enforcement Rules

Security and trust are two independent variables, which require two different types of enforcement. Anonymized data with a TL=2 may be sent with a SL=1 in plaintext (e.g. LHW scenario), while data of TL=1 may be sent with a SL=2 because the driver wants to keep them private. Therefore the STL is defined as the concatenation of the SL and

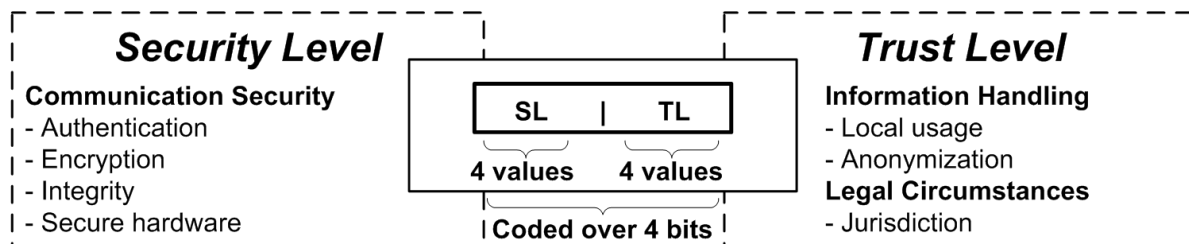


Figure 3.7: STL vector and main evaluation criteria.

the TL, as shown in Figure 3.7. For an efficient enforcement, the STL is limited to 4 values for the SL and 4 for the TL and can code the resulting vector over 4 bits.

Concretely, data arriving to the proxy with a $STL_{REQ}=(sl_{req},tl_{req})$ will be allowed to be released to an external service or device X assigned a $STL_{STATUS}=(sl_x,tl_x)$: 1) if X complies with the conditions of the received TL, i. e., if $tl_{req} \leq tl_x$ and 2) if the communication with X with the conditions of the received SL, i. e., if $sl_{req} \leq sl_x$. However, such conditions may be too constraining and may never allow certain data to leave the car. Declassification methods allowing to assign a lower STL to some data or to add just an exception on the proxy should be possible. But those methods should only be part of use cases predefined by car manufacturers and if necessary should involve the driver's decision, e. g., if it is her private data. Further considerations about declassification methods are not provided in this work.

STL-based policies are statically implemented in the ECUs and have to be evaluated on a case-by-case basis. They do not require any regular update. Either the ECU generates the data to be sent and associates its own STL_{REQ} depending on the appropriate policy, or the ECU received the data from another ECU and before forwarding them, labels them with the received STL_{REQ} . The proxy should regularly receive notifications to update the TL of new external services and the SL of new or flawed communication protocols. The CE device case is a little bit particular. But since this device gets authenticated by the proxy, a STL_{STATUS} is assigned to it. This STL depends on the used connection protocol for the SL and is assigned a $TL=1$, since it is assumed that the user's device is under her control and is therefore safely handling her own private data.

3.3 Middleware and Security Discussion

This Section concludes this chapter and provides a discussion about the SME architecture in Subsection 3.3.1 and about the security communication proxy in Subsection 3.3.2. In addition to listing the pros and cons of the two architectures, this section also proposes three security configurations for the SME setup. Then, Subsection 3.3.3 pinpoints the remaining issues and limitations of the SME/STL approach and the challenges it does not tackle yet. Finally, Subsection 3.3.4 evaluates the middleware security reaction

for concrete attacks on the use case of Section 2.3.3.

3.3.1 About the SME Architecture

Implementing security at the middleware level allows to follow an engineering-driven software development. Security and middleware developers may remain unaware about what the applications are really doing. They only need to have a superficial insight of this layer, i. e., the application purpose, some requirements and a clear definition of the communication interfaces. Instead of designing new security mechanisms, the SME leverages current security protocols and technologies from the Internet and CE world and optimizes their usage to fit to the requirements of the automotive world.

More than only providing access control mechanisms and secure communications, the security middleware layer can be considered as a protective barrier for all automotive applications. Developing a single middleware for the whole car allows car manufacturers to apply security programming guide lines to avoid security attacks, like buffer overflows [64, 79, 118] and reduce the amount of code to verify and maintain. In addition to these development approaches, methods for software security assurance evaluation can be used to formalize penetration tests and anti-requirements, i. e., implementation to avoid [93, 95, 92]. In a next future, model-based design approaches with formalized verification models will automate these tasks and significantly reduce the integration effort [42]. For the moment, they are mostly used for verifying the correctness of transition system at the functional level (e. g., fault-tolerance), but could also be extended for security in order to automatically patch any automotive software [30].

Then, the SME benefits from a flexible and modular design allowing the different SME versions to be interoperable and adaptable to most automotive platforms and use cases. At a functional level, the SMEs can secure the three middleware versions designed by Weckemann et al [178]. In addition, they leverage previous work on hardware security, e. g., the three-level HSMs of EVITA [182]. Table 3.8 presents in more details the three SME levels and their specifications.

Nonetheless adding security should neither degrade the car performance nor cause additional faults in the functional middleware and its applications. This work does not aim at investigating the impact of faults on safety or security. The following paragraph just briefly presents some issues and provides a few research approaches. Faults are defined as any interruption of the normal operativeness of a system or of any of its sub-components. They include both accidental and intentional ones, e. g., an attack. Numerous safety mechanisms allow to cope with accidental faults, e. g., high redundancy, fall-back mechanisms, error detection or self testing. However intentional faults are harder to detect, differentiate and isolate. When a fault occurs, the fault detection mechanisms should be able to identify it, create a fault awareness and restore the service. This paragraph does not propose new mechanisms, but shortly discusses the reaction scenario of an automotive Fail Safe Mode (FSM). In case of a malfunctioning security service, such process should be launched only if the car is unable to restore all

Table 3.8: SME specifications of the three-level middleware approach (MAXIMUM , MEDIUM and MINIMUM Security)

SME security level	MAXIMUM Security	MEDIUM Security	MINIMUM Security
GSM & KMM	<ul style="list-style-type: none"> - EVITA high HSM [182] - IPsec channel management with preshared key and dynamic key establishment via IKEv2 	<ul style="list-style-type: none"> - EVITA medium HSM [182] - IPsec channel management with preshared key and dynamic key establishment via IKEv2 	<ul style="list-style-type: none"> - EVITA light HSM [182] - IPsec channel management
SCM & ANM	<ul style="list-style-type: none"> - VPN tunnel management 		
(e.g., remote flashing and diagnosis)			
PMM	<ul style="list-style-type: none"> - middleware-based policies - application-based policies - host-based IDS 	<ul style="list-style-type: none"> - middleware-based policies - application-based policies - host-based IDS 	<ul style="list-style-type: none"> - middleware-based policies - no IDS
IDM	<ul style="list-style-type: none"> - network-based IDS - introspection-based IDS 	<ul style="list-style-type: none"> - network-based IDS 	
Examples of ECUs	<ul style="list-style-type: none"> - HU - C2X Gateway 	<ul style="list-style-type: none"> - engine control - ESP control 	<ul style="list-style-type: none"> - sensors - brake actuator
ECUs		<ul style="list-style-type: none"> - central routing gateway 	<ul style="list-style-type: none"> - airbag actuator

necessary security services to an acceptable level of functionality. A complete reboot or deactivation of the system is not feasible, the car should always be maneuverable and able to restart the engine even when being in FSM. The FSM measures can be summarized in two classes of actions that should be performed simultaneously. First, at a security level, the car should be able to shut down completely or partly the buggy security service, e. g., stop encryption between some ECUs or bypass some access control mechanisms. Secondly, at a functional level, it should be able to adapt other non-faulty services in order to limit the impact of the error and the newly unprotected surface of attack, e. g., by deactivating all C2X communication interfaces, limiting the driving speed or the engine revolutions per minute and informing the drivers about the error status via audio or visual signals. The FSM should be used as last resort since an attack could aim at leveraging it in order to disable some security features or limit the car functionality and performance.

The SME architecture and related recommendations aim at limiting errors and their propagation. The focus was to reduce the presence of Single Point of Failure (SPF), which increases the risk of high latency and system blocking. Every ECU is independent of its security decisions and is capable of establishing its own communication channels during the car runtime. However, for cost reasons and to avoid components redundancy, the SME includes some central interfaces, e. g., *Security Master ECU*, but only for non time-critical functionalities, like rekeying, policy updates or IDS mechanisms.

3.3.2 About the Security Proxy Architecture

The communication decoupling at the proxy level offers several advantages. First the Third-Party (TP) developers can remain security-unaware and are free to choose their own communication protocols, as long as the proxy or the on-board applications are compatible with them. Secondly, on-board communications occur on a unique set of middleware based on strong security protocols. Thirdly the middleware protocol can be extended with a STL field in order to support holistic security decisions at the ECU level. However, even if the C2X communication traffic will generally remain small in comparison to the total on-board traffic, the proxy is a unique interface for all C2X use cases and could act as a bottleneck. Further tests on actual implementations should be performed in order to confirm or abnegate this consideration.

Then, ECUs and proxy rely on each other's integrity for delivering of valid and accurate STL. An attacker may want to bypass or corrupt the proxy and tamper the STL process. The malicious messages would be assigned the STL of a trustworthy situation and would get access to more functionalities. Current HSM architectures only include secure boot mechanisms and would not be able to counteract runtime attack. The proxy IDM here plays an essential role. The integration of hypervisor or microkernel architectures [90] could provide isolation and monitoring between the different C2X communication interfaces of the proxy. Each group of communications (e. g., Wi-Fi-, 3G- or LTE-based) in one cell disposes of their own security mechanisms and may be isolated

from each other. As a result, despite a successful attack, the proxy can shut down the compromised cell, potentially restart it and still be functional. Further investigations, not performed in this work, need to be done in order to determine the suitability of the approach.

Finally, several use cases involving automotive software downloads and on-board firmware updates may require end-to-end security solutions, e.g., activation of the ECU reprogramming mode. For such use cases, the proxy should be able to provide secure tunneling, e.g., VPN-like, between a server of the car manufacturer and the targeted ECU. The infrastructures of the proxy and of these ECUs (e.g., HU) require to be designed and adapted accordingly.

3.3.3 About the STL approach

The enforcement of STL rules in the middleware is relatively simple to manage and can be efficiently performed. Nonetheless it requires that car manufacturers determine the authorized STLs for every single communication interface. This STL definition can certainly be easily described and integrated to the middleware development thanks to the IDL. But the STL remains mostly focused on the release of information to the outside world and only considers whether the messages can leave the car, whether they include private data and on what kind of C2X security channels they can be transmitted. Analyzing and verifying on-board STL-based information flows may be quite complex and not very relevant for exclusively-on-board communications. Additionally, the STL considers no information integrity but just its confidentiality. Besides, the system implicitly considers a unique user, i.e., the driver. Adding to the STL the ID of a user or of the information source may not be sufficient without a proper formalization.

Then, for the moment, the SME/proxy architecture does not consider any solution for integrating Third-Party Applications (TPAs). Additional mechanisms should be added to the car communication infrastructure in order to constrain TPAs to respect the authorized on-board information flows defined by car manufacturers and to not disturb the overall car functioning. Finally this chapter does not propose any formal evaluation of the STL taxonomy or a clear integration of the user preferences. Instead, the goal was to describe concrete and easily-enforceable examples of security and trust levels based on clear security requirements and quantitative parameters.

3.3.4 Security Gains

This section evaluates the security gains of this architecture in the context of the use case presented in Section 2.3.3. A few network and middleware attacks are listed here. For each of them, a short description about how the system reacts to the threat is then provided.

Eavesdropping: An attacker with physical access to the on-board network can listen to the on-board communications and gain access to sensitive exchanged information,

e. g., driver's private data being exchanged between proxy and HU.

Security measures: Via their SCM, ECUs establish secure channels leveraging strong encryption (e. g., AES encryption with IPsec) and preventing an attacker without knowledge of the keys to decipher the communication. The encryption keys are assumed to be stored in a tamper-resistant HSM and therefore out of reach of the attacker considered here.

Result: This attack cannot be leveraged and harm the car, its passenger or the car manufacturer.

Replay attack: An attacker with physical access to the on-board network can reinject a message earlier sent, e. g., to bypass the VSS and send a null speed to the HU and watch TV or open the convertible roof while driving.

Security measures: Via the SCM, ECUs establish secure channel leveraging nonces for freshness verification and preventing an attacker from replaying a message. Besides if detected, the SCM can notify the IDM for suspicion of replay.

Result: This attack can easily be detected and cannot cause any harm to the car, its passenger or the car manufacturer anymore.

Addition of a new network node: An attacker with physical access to the on-board network could plug a new ECU, e. g., a new HU, get access to more features and potentially make the whole system unstable.

Security measures: ECUs via their SCM establish secure channels approved by their PMM. The communication from a new ECU with an unknown set of cryptographic material would be dropped by their SCM, because it was not recognized by their PMM. Besides if detected, the SCM can notify its IDM for invalid communication.

Result: This attack can easily be detected and cannot cause any harm to the car, its passenger or the car manufacturer anymore.

ECU corruption: All ECU hardware attacks, i. e., physical manipulations, are assumed as out of scope. Therefore attacks covered by this work have to be network based and sent to a communication interface of the ECU.

Security measures: Developing a middleware-based software architecture reduces the amount of code to verify against stack pointer overwriting attacks like buffer overflows [4]. Secure coding guidelines can be specified and reduce the risk of such attacks. Besides the use of a HSM and its secure boot allows to detect any memory modification (e. g., invalid running code) but cannot be used during runtime. The only solution is to rely on host-based IDMs, however due to the performance requirements they cannot be setup everywhere in the car.

Result: The attack is alleviated but cannot be completely stopped.

Denial of service attack: An attacker with access to the on-board network could launch a DoS attack, e. g., jamming the network to prevent communications between ABS and VSS or sending multiple requests to the PCM in order to isolate it from the network or from a TPA installed on the HU consuming all resources of its host and making it ineffective.

Security measures: Network-based DoS attacks can easily be detected by strategically-placed IDMs monitoring the network. However, at this point of the thesis, a DoS attack launched by the TPA cannot be fought back.

Result: This attack cannot be successful at the network level anymore but can still succeed if performed by a TPA or on a ECU without host-based IDM.

Unauthorized function/data access: 1) An attacker with remote access (via the proxy) to the on-board network could subscribe to an on-board multicasting address in order to have the vehicle GPS information forwarded to an untrusted online server. 2) A attacker could program its own TPA, have it installed on the HU, leverage the HU secure communication channels and leak GPS information or disable the ABS.

Security measures: GPS information is labeled as private by the GPS module itself. If an attacker manages to have it forwarded, the proxy would recognize the label and stop it before leaving the car. At this point of the thesis, such a TPA-based attack cannot be fought back.

Result: Leakage of private data to an external untrusted entity can be mitigated, except if the connected CE device is compromised and leak information. Same as before, TPA-based attacks are not considered yet.

This section shows that the integration of secure communication protocols, efficient policy management and HSM can counteract numerous attacks presented in the related work of Chapter 2. Software corruption and DoS attacks can be mitigated by security mechanisms integrated within the middleware. Attacks related to unauthorized resource access and TPA integration have to be deeper investigated in the next chapter.

3.4 Summary

This chapter presented a new middleware-based security architecture, namely the SME, which leverages the new bandwidth and computation capacity of next-generation on-board communication networks and enhances its security. The goals of the SME are twofold: establishing security channels and providing suitable in-car access control mechanisms. Working at the middleware level allows to abstract all security considerations from the application logic and to develop a secure software layer based on security and engineering-driven guidelines.

Regarding the C2X communications and integration of online services, the communication decoupling performed at the proxy level allows to internally preserve the security and communication homogeneity. Third-Party developers of CE-based applications may remain security-unaware, the proxy is in charge of assessing the security and trust of the communication and supports the ECUs for the enforcement of appropriate security decisions. The extension of the in-band middleware protocol gives the opportunity to cars to enforce a STL-based approach for IFC. The STL is described in details in this chapter and enables information releases respecting the confidentiality of both passengers and car manufacturers.

Information Flow Control in Cars

While providing efficient management for the establishment of secure communication channels, the SME and the security communication proxy rely on quite simple access control mechanisms that may be insufficient to mitigate the security and privacy risks related to C2X communications. Besides the integration of Third-Party Applications (TPAs) results in new on-board security threats, that the on-board infrastructure should also consider.

This chapter presents an efficient and scalable security model providing Information Flow Control (IFC) and access control at different levels: (1) at the network level in order to efficiently control communications and information flows between on-board applications developed by car manufacturers (2) at the application level in order to deeply monitor TPAs and fully control their execution environment. Like in Chapter 3, the middleware leverages the large bandwidth capacity of Ethernet/IP and serves as glue to hold all security features together: across the on-board network and across the different local levels of security enforcement.

First, Section 4.1 introduces in details a first formal IFC approach of this thesis, which leverages a new automotive Decentralized Information Flow Control (DIFC) model coupled to an isolation environment hosting the TPAs. Then, Section 4.2 describes and explains a second approach making use of the STL-based model of Chapter 3 and Dynamic Data Flow Tracking (DDFT) techniques for a full control over the TPA. Finally, Section 4.3 proposes a third IFC approach, which combines the two solutions developed in the two previous sections, i. e., the combination of the DIFC model and the DDFT techniques. The two first approaches are independent and present their own related work and discussions; the last one proposes a comparative discussion and provides architecture recommendations from a pure security point of view.

Parts of this chapter were previously published in *Middleware-Based Security and Privacy for In-car Integration of Third-Party Applications* [26], *Practical Information-Flow Aware Middleware for In-Car Communication* [29], *Leveraging In-Car Security by Combining Information Flow Monitoring Techniques* [28], and *Middleware-based Security for*

Hyperconnected Applications in Future In-Car Networks [23].

4.1 Decentralized Information Flow Control (DIFC)

Focusing on the security of the communication link, i. e., on authentication or encryption and simple ACLs, will not solve all information security issues of distributed systems like cars. The range of action of these mechanisms is limited to communications between two on-board platforms, i. e., two ECUs. Thus they do not provide any solution against mistakes at the application layer or untrusted communication partners, e. g., CE devices or TPAs abusing from their capacities to leak sensitive information or misuse some on-board resources. On the other hand, the staticity of the communication channels allows car manufacturers to partly know on beforehand a big part of the traffic generated by the ECUs. This knowledge can help them to define abstract zones including whole ECUs or some applications and in which pieces of information from same sensitivity may flow. The traffic may therefore be labeled depending on its sensitivity. Such abstraction allows to delegate the design of the security management to a team of security experts. These experts may only have a superficial understanding of the application, i. e., what information comes in and out and can really fully focus on the management of this security zones.

In order to formalize these zones and traffic labeling, this first approach rely on DIFC. DIFC provides methods and rules to control which pieces of information can be exchanged and how they spread across the on-board network. The middleware is also considered as a trusted computing base in comparison to the potentially flawed applications running over it. This software layer allows the car to act as a data safe by regulating how the information is accessed and leaves the on-board network.

After having given an overview about traditional IFC approaches and related work in Subsection 4.1.1, Subsection 4.1.2 introduces a DIFC model for automotive environment. Then, Subsection 4.1.3 presents how such a model is integrated to the SME and to the security communication proxy. Afterwards, Subsection 4.1.4 proposes a first discussion about the solution advantages and disadvantages.

4.1.1 DIFC Related Work

IFC is not a new topic. Already in the 1960s, Lampson was demonstrating the insufficiency of only using access control mechanisms to protect sensitive data [105]. Information leakage occurs when a first entity A is authorized to access an object of a second entity B and discloses it to a third one C, whereas C does not have the object authorization access from B. This confinement issue can be extended to the integrity level, when A modifies a resource of B on behalf of C, even though C is not authorized to do so.

Originally used for the context of military multi-level systems [171], IFC is a type of mandatory access control, which attempts to solve these issues. Principals, i. e.,

persons, and objects, i. e., documents are assigned labels characterizing a sensitivity level. The access decisions are regulated by a “can flow to” partial order. For example, a principal labeled as “secret” can read a document of lower level like “confidential” but not of a higher level like “top secret” (simple security property of the Bell-Lapadula model [12]) and cannot write on a document of lower level (\star -property [12]). This last model is focused on protecting the information flow confidentiality, but others like the Biba model [17] can insist on the integrity protection. In a similar way, an IFC-protected object can be seen as having a contaminating or tainting effect, i. e., a principal having access to a labeled object must be assigned the same security label and will transmit it to all other objects it accesses like a contamination. However, this compartmentalization of a system in label-based zones still remains coarse-grained and requires to have an object declassification feature, i. e., a possibility to modify labels that should occur in highly-trusted centralized units.

DIFC has been first introduced by Myers and Liskov in 1990s and allows a discretionary control of policy decisions delegated to each principals and objects and not relying on any central administration [130]. For example, every automotive application and middleware could be in charge of their own labels and administrate them during runtime in order to protect the user’s and car manufacturer’s policy as well as the car integrity. The rest of this subsection focuses on two major lines of research related to the enforcement of DIFC: (1) programming languages and (2) operating systems.

4.1.1.1 Programming Languages

Jflow [128] and its successor Jif [129] are java-based programming languages implementing a DIFC model. They provide program annotations expressing data security labels and make use of custom compilers to track and enforce information flows within a program. They rely on static code analysis. Variables are labeled based on their owners (i. e., some principals) and other principals can overwrite the labels, i. e., declassify and modify them, only if all owners allowed it. Originally they do not provide any interaction with the OS, e. g., for file or socket management and perform on a closed-world assumptions, which can be invalidated if the program use dynamic extensions [67]. These languages are only designed for a single sequential program without concurrency or dynamic label update.

Following Jif-based approaches extended the language to make it fit some requirements of real-world applications. The rest of the paragraph does not provide an exhaustive list of Jif-based extensions, just some of the most relevant ones. SIESTA [70] allows to combine Jif and SELinux [117] to label files and sockets as well and use the mandatory access control functionality of SELinux directly from the application level. Several additional extensions provide methods to dynamically update the labels hierarchy [71] or methods to add new labels and principals after compilation time [170]. The SIF framework allows to build Jif-based web applications, where labeled applications are mapped as principals, can get authenticated and establish secure sessions [38]. Additional work

on robust declassification methods prevents principals from retrieving information likely to be used to influence the IFC system [37].

Outcome: The development of automotive applications should not involve any custom programming language and add more complexity for the application developer like security annotations in the source code. Additionally the approach, earlier explained in this chapter, states a security at a coarser level for information flow, e. g., between application blocks, and not at the variable or object level. Besides these languages cannot be used to secure TPA, because by definition, TP developers are untrusted and do not program in a secure and reliable way.

4.1.1.2 Operating Systems

In comparison to programming languages, OSs rather manage tags instead of policies linked to principals. Tags identify groups of principals and allow to have just a few policies managing all tags, instead of one per principal. Then, they perform dynamic checking instead of static analysis. As the system runs, OSs determine the accurate status of an information flow, but at a coarser level, since it is not possible to really let OSs label all variables like previously. These OSs work with process labels (including the tags), which can be stored for later use and which abstract the notion of principal from the model. Finally they allow to use process labels for IFC but also for resource access control.

Asbestos [52] is a UNIX-based OS enforcing DIFC at the process level, labels are used to express the contamination of the process and its set of tags. Asbestos provides a way to grant these tags between processes but does not address their revocation. HiStar [190] follows the same model but at the kernel-level. As a consequence the secure base is much smaller and the user-level can be untrusted. In HiStar, all threads request secure labels changes and perform all operation in their own address space. IPCs are performed through “gates” which protect control transfer and resource allocation to avoid covert channel attacks. DStar [191] extends HiStar over the network, controls distributed resource allocations to avoid covert channels and exchanges tags through the network for trust delegation. Security is ensured by a system of cryptographic certificates for tag delegation and easy revocation. Pedigree [149] proposes a trusted kernel module for enforcement of IFC policies on legacy applications. The module is hooked to relevant system call of the host, the enforcement is enforced either on application hosts or on custom switches using a custom Ethernet-based protocol. The whole architecture relies on a central server for policy synchronization.

Flume [104] is implemented at the user-level and not influenced by different OS versioning. Flume provides simple intuitive labels (i. e., secrecy, integrity tags and ownership) and keeps track of the application labels in a central tag registry. Laminar [153] is based on Flume and combine programming languages (i. e., a modified Java Virtual Machine (JVM)) and kernel extensions. It supports process sharing objects. Programs using Laminar include label-assigned parts of code called “security region”. DIFC rules

are enforced when objects are exchanged between two security regions. DEFCon [122] is based on a modified JVM enforcing the Flume model. DEFCon blacklists part of the Java API and proposes a new API allowing to protect objects between competitive units embedded in the JVM.

Outcome: In contrast to most of these approaches, automotive software components are distributed over several hardware platforms equipped with different OSs. In order to reduce the risk of errors, the latency and maintenance complexity, the IFC cannot rely on any central entity and cannot be directly enforced by the hardware or its OS. As a result, the security will be enforced at the application level and will not be able to control the file access or socket management. But at the same time, official developers of car manufacturers can be trusted to only access authorized files and authorized middleware API. Therefore it may still be reasonable to only enforce the access control within the middleware in order to control information flows between application blocks.

4.1.2 DIFC Model

DIFC is about monitoring the propagation of data of interest. The automotive approach performs its monitoring on information flows, i. e., communications between on-board applications via the middleware. Obviously several applications run on top of a same middleware. For an easier security management, applications sharing the same security concerns are regrouped on a same middleware layer. The resulting group of applications and middleware is called *service*. Applications can share the same concerns in terms of (1) confidentiality, because they process, exchange data of same sensitivity and (2) integrity, because they trigger the same mechanisms or modify the same resources. For this reason, like processes for the DIFC-enabled OS, each service is assigned a label expressing their security concerns. The middleware, which is independent from the potentially flawed applications and can be easily verified, is in charge of enforcing DIFC with these labels. The rest of this subsection describes an automotive DIFC model, adapted from [191, 104].

4.1.2.1 Security Labels

One security label is assigned to each principal, e. g., a service. Comparing labels of two services allows to constrain the information flows between them and therefore to protect the information integrity and confidentiality, for example by isolating potentially corrupted data from critical applications or preventing unauthorized disclosure of private information.

Labels consist of two components: the first characterizing the principal's secrecy S , the second its integrity I . S and I are two sets of tags. A tag represents the concern of an individual about the secrecy/privacy (in S) and integrity (in I) of some data. Tags are unique values in the system, implemented as bit-strings, they are called with symbolic names, like x_s . The subscripts s and i of x_s and x_i designate, respectively a

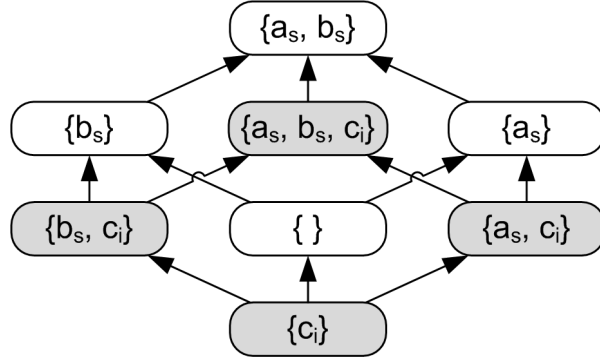


Figure 4.1: Label-based lattice. This example includes 2 secrecy tags a_s and b_s and one integrity tag c_i . Boxes represent service labels. Grey boxes show labels including c_i . Arrows link labels between which information can flow.

secrecy- and an integrity-tag. The x specifies the principal whose security concerns are characterized, i. e., the service x or the CE device of the driver x . The notion *service tag* and *user tag* designate tags characterizing the security concerns of respectively an on-board service or a user and her CE device. Secrecy tags are “sticky”: once added to a piece of information, it cannot flow to a principal lacking the exact same tag. On the other hand integrity tags are fragile: a piece of information loses it as soon as the principal processing it is differently tagged.

As shown in figure 4.1, information flows between labeled principals can be represented in a lattice following a form of mandatory access control. Information from a principal labeled with the secrecy tags of S_A can flow to a second principal labeled with S_B if and only if the tags of S_A are included in S_B . Inversely, information from a principal labeled with the integrity tags of I_A can flow to a second principal labeled with I_B if and only if the tags of I_A contain the ones of I_B . The partial order “ \prec ” for labels (pronounced “can flow to”) can be formally defined as:

$$L_A \prec L_B \text{ iff } S_A \subseteq S_B \text{ and } I_A \supseteq I_B,$$

$$\text{where } L_A = (S_A, I_A) \text{ and } L_B = (S_B, I_B)$$

Because the services are distributed over different ECUs and do not necessarily know each other’s labels, the exchanged messages are labeled as well. When A sends a message M to B with L_A, L_M, L_B their respective labels, the property $L_A \prec L_M \prec L_B$ is enforced by the middleware of A and B . Constraining the message label allows the message to be disclosed by A (i. e., A can label M with L_A such that $L_A \prec L_M$) and then to be accepted by B (i. e., B checks the condition $L_M \prec L_B$ and thus $L_A \prec L_B$). For an automotive scenario, data stored on the HU should be tagged with different values reflecting the different drivers’ secrecy, so that only appropriate TPAs or CE devices can receive a message containing a particular driver’s data.

4.1.2.2 Tag Ownership

If information could only follow the partial order “ \prec ”, the labeled messages would only be transmitted to principals classified at an equal or higher level of secrecy and most data would never be able to leave the car. DIFC decentralizes the management of exceptions: each service S may be assigned a set of tags O , allowing it to derogate from the label restrictions included in O . S is told to own the tag of O . Obviously no service should own all the tags of the system; a service should own only the tags necessary to remain functional.

A new partial order “ \prec_O ” (pronounced “can flow to, given O ”) taking into account the ownership to the tag of O can then be defined. Practically, a tag t included in O confers the possibility for a service S to omit the restriction imposed by t . For data flowing from A to B , except for the tags included in O , the label L_A contains all the integrity tags of L_B and L_B contains all the secrecy tags of L_A . “ \prec_O ” is formally defined as:

$$L_A \prec_O L_B \text{ iff } S_A - O \subseteq S_B - O \text{ and } I_A - O \supseteq I_B - O,$$

$$\text{where } L_A = (S_A, I_A) \text{ and } L_B = (S_B, I_B)$$

Like earlier, A with the ownership O_A can send a message M and B with the ownership O_B can receive it if and only if $L_A \prec_{O_A} L_M \prec_{O_B} L_B$. An untrustworthy TPA will not be given any ownership and therefore will not be able to modify its own label in order to leak the driver’s data or access other driver’s data. On the contrary the proxy will be given the ownership of all driver’s tags in order to be able to send a driver’s data to her CE device. The proxy provides a high security level and is trusted to use its ownership in a secure manner.

In order to express a new security concern, during runtime a service can create and own a new tag. At its discretion, it can grant the ownership to other services. For each new user U , the proxy generates a new secrecy tag u_s and grants it to the HU, so that the HU can label and protect the private data of U .

4.1.2.3 Dynamic Label Assignment

A Dynamic Label Assignment (DLA) is an explicit request from a service A to another service B to increase the label of B with a new tag. The DIFC approach in this section considers TPAs as being enclosed in isolated cells on the HU. Like for a “black box”, the HU can only monitor the inputs and outputs of the isolated cell and therefore of the TPA. At first the TPA is empty labeled without any ownership and thus cannot receive any sensitive, i.e., secrecy-labeled, information or contact integrity-critical functions. For example, in order to exchange private data of the driver d , the HU, which owns d_s , imposes per DLA the TPA to extend its label with d_s . The TPA cannot take the tag d_s out of its label later and is therefore only able to send messages to services including d_s in their label or ownership. The TPA is label unaware and does not manage its own

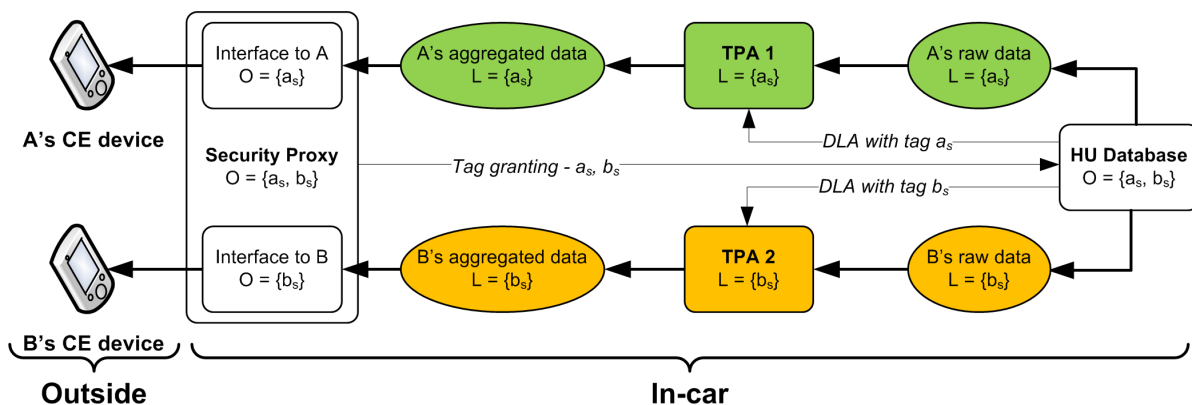


Figure 4.2: Example of label usage preventing a TPA to inappropriately leaking information of 2 users A and B. Ellipses are on-board labeled messages. Colored boxes are labeled components with the user's tag a_s and b_s of respectively user A and B. The proxy authenticates the CE devices, creates the new tags and a_s and b_s and communicates with the HU database to grant it their ownership. In order to send to the TPA a user's raw data, the HU database performs a DLA extending the label of the TPA.

label. Instead, a trusted dedicated service of the HU is in charge of it and filters all inputs/outputs of that TPA.

For the purpose of this paragraph, Figure 4.2 is kept simple and illustrates an example of how to use labels to protect the confidentiality of two users. The proxy owns the two tags a_s and b_s but only maps one tag to each communication interface. The two TPAs are obviously running in two environment cells isolated from each other. Security here partly relies on a good isolation of the cells. More information about these mechanisms is provided in Section 5.4.1.

Reselling and scrapping the car are part of the vehicle life cycle and should also be taken into considerations. DIFC can support these phases. The HU owns the right on the private data stored in the car and could erase all sensitive data. Car manufacturers could for example develop a procedure leveraging this point that could be performed by mechanic approved by the car manufacturer. Such a solution is not further discussed in this work.

4.1.3 DIFC-enabled Middleware

In order to provide suitable performance and to limit the risk of error, the DIFC monitoring is only applied to on-board information flows between services. Applications in a same service share the same security concerns and therefore can be labeled together. Services are isolated from each other in their own address space or physically separated, i. e., on different ECUs. Enforcement of DIFC at the middleware level is indeed more

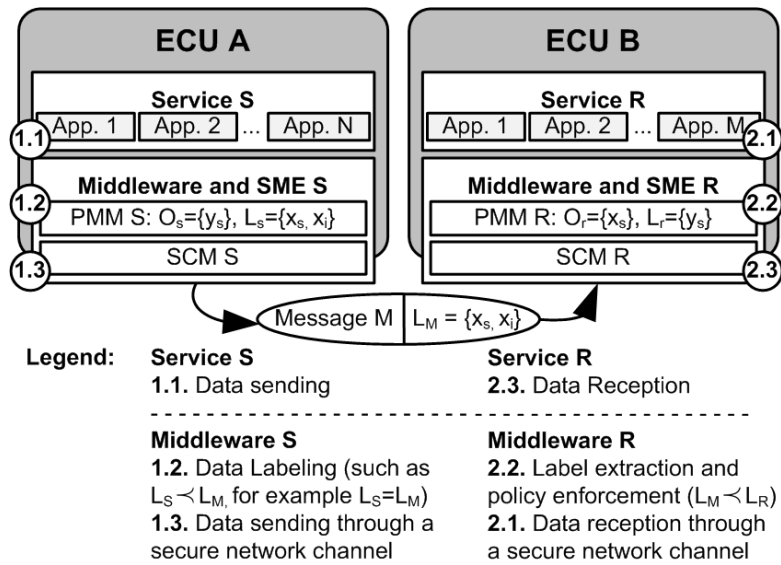


Figure 4.3: Overview of the DIFC-enabled middleware architecture. A labeled message M is exchanged between 2 applications (App.) of S and R services through a secure channel. L and O designate the service label and ownership. x_s , y_s and x_i are secrecy and integrity tags. (SME: Security Middleware Extension, PMM: Policy Management Module, SCM: Secure Channel Module,)

suitable: this software layer is common to every service, easily auditable and in charge of the network communications. The architecture of the DIFC-enabled middleware is depicted in Figure 4.3, for more simplicity the SME architecture has been simplified and only focuses on the relevant modules. Applications in different services interact through their middleware, which provides the functional logic for communication and protocol implementation (SCM). The middleware header of every message is extended with a field containing the message label, e. g., with the label of service S . The SCM makes sure that the communication channel is valid and that the partial order \prec_O is respected. For this second task, the SCM is supported by the PMM which is aware of the service label and ownership and can provide the right DIFC policy decision for both incoming and outgoing traffics, e. g., the middleware of service R enforcing $L_M \prec_{O_R} L_R$. Applications are DIFC-unaware and do not take part of the label management. The remainder of this section provides more specifications about the automotive label assignment, policies and management.

4.1.3.1 Label Assignment

Each service x is labeled with its own integrity and secrecy tags (x_i , x_s) characterizing its own security concern. The assignment of additional label tags or tag ownership is defined by car manufacturers at design time and depends on the use cases the service is

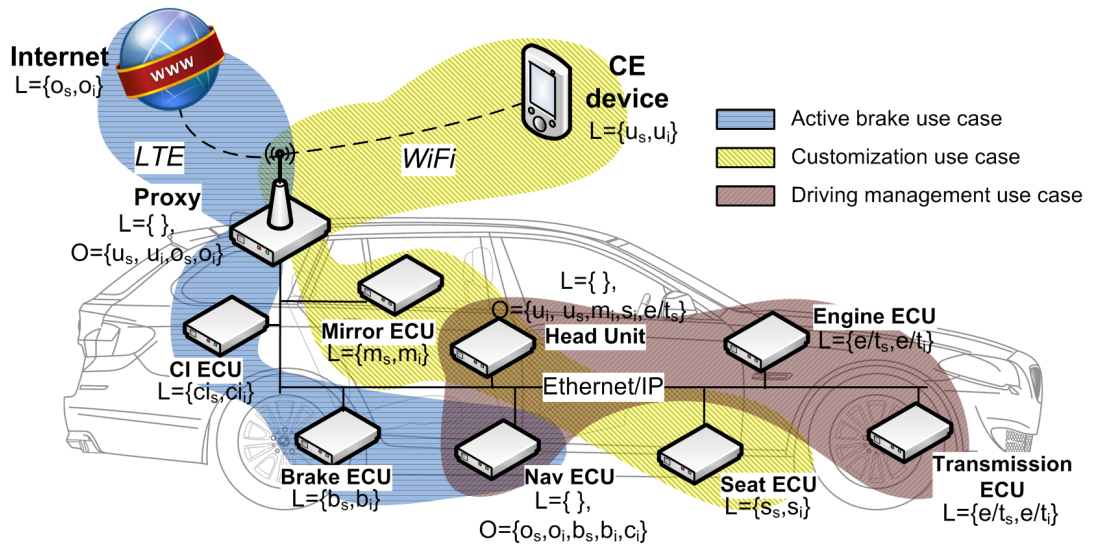


Figure 4.4: Example of on-board label distribution over 3 use cases. The labels include secrecy and integrity labels. For more simplicity, the services are not shown and ECUs are assumed to run one service each. The naming of the label follows the ECU abbreviation, whose security concern they express. The ownership allows the Nav(igation) ECU, HU and proxy to take part in several use cases.

involved in. During runtime, the proxy is the only one able to create new tags related to a new user profile and to grant them to relevant ECUs, like the HU. The addition of new services is not considered. Therefore no new service tag is generated during runtime, i. e., after the car left the assembly line. But obviously, new service tags would force the car maker to update the middleware label and ownership of every middleware communicating with the new service.

A tag distribution was applied as example on three use cases and can be found in Figure 4.4. The use cases concern the infotainment (customization), the safety (active brake) and the engine control (driving management) purposes. They present different levels of sensitivity, handling private information or triggering critical actions. A suitable labeling allows to isolate the information flows in distinct areas not disturbing and compromising each other. A specific configuration of the ownership label in trusted ECUs also allows these information flows to be processed by ECUs and applications involved in several of these use cases.

A second tag distribution approach is to follow the domain-based architecture of the car presented in Section 2.2.1. Each service of a domain X gets assigned the tag X for secrecy and integrity. Secure services on the edges of these domains, i. e., communicating with other domains are given the ownership of the tags they require. As a consequence simple services, i. e., without ownership, can freely communicate within a domain and

go through secure services with appropriate ownership in order to reach services in another domain. The secure services perform their declassification privileges based on their ownership according to security policies predefined by car manufacturers. While being extremely simple, this approach may not provide the expected fine-grained security enforcement, especially during complex use cases and is therefore not recommended.

4.1.3.2 Label Policies

In addition to the application policies presented in Chapter 3, a new type of middleware policy can be specified. These policies are still enforced in the SCM and evaluated in the PMM but now carry DIFC information as well. As mentioned in the subsection 4.1.1, DIFC allows to limit the number of policies and their complexity by enforcing logical relationships on the labels. Depending on the service label assignment, the policies now enforce the partial orders \prec or \prec_O . Practically they specify which labels are authorized for incoming messages, and which label to add to every outgoing message. Defined by the security experts, the label policies are defined during the design phase and remain static during runtime.

4.1.3.3 Label Management

On-board services exchange messages for various purposes and may store them along with their DIFC label. For example, a tracking application on the HU may frequently request GPS coordinates and car status information, storing these data and their secrecy label in order to protect the concerned driver's privacy. However, in case of complex data fusion involving different labels from users and services, managing the resulting label may be problematic. Concatenating all labels together may lead to message that may be refused by every on-board service. Instead prioritization rules have to be defined, in order to keep one secrecy tag and one confidentiality tag for every message. The labels are kept simple and easy to be processed by the SME. Prioritization rules depend on the tags and the relevancy of the security concern they have to protect. This paragraph only gives an idea about how they should be designed. For example, a secrecy tag of a service, handling sensitive information of the car manufacturer, should have a higher priority than a user secrecy tag, because sensitive data for the car manufacturer should stay in the car and not be disclosed to any user. These rules mostly concern the secrecy tags, the integrity tags are "fragile" and dependent on the service and middleware producing the data, i.e., realizing the data fusion. The choice of labeling the data with its own service tag is specific to every service and its capacity of producing safe data respecting a certain level of integrity. The prioritization rules are not expected to be extensively used, only for car customization use cases, e.g., statistics about a driver's driving, user's voice recording, picture management.

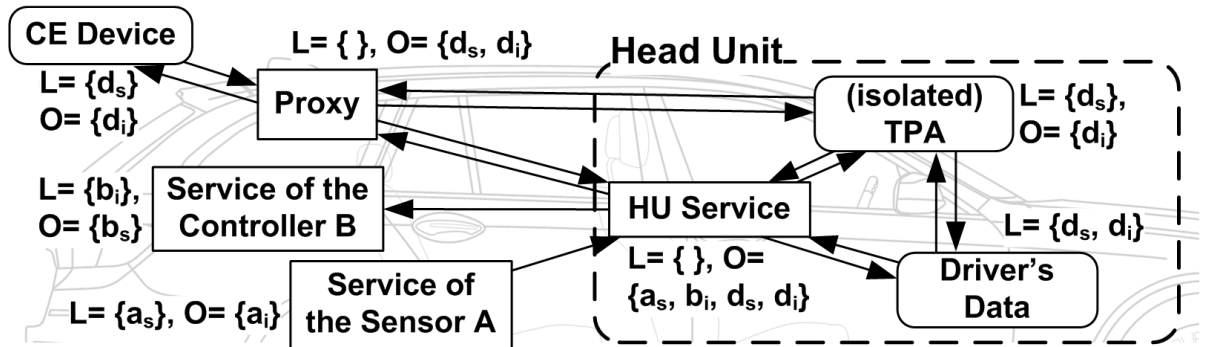


Figure 4.5: DIFC-enabled automotive scenario. Rectangles represent services running on an independent middleware. Round boxes represent DIFC unaware applications, devices and files. Solid arrows represent authorized middleware-based communications.

4.1.4 Discussion

In order to illustrate how the DIFC approach helps to build a secure on-board system, this subsection focuses on the scenario presented in Figure 4.5. The label distribution of this scenario has been established, so that the following on board communications can be authorized. The proxy may simultaneously communicate with a HU Service, the TPA and some external devices not part of DIFC-protected system; that is why it is empty labeled and owns all the tags it needs. Then the HU Service may communicate with first a service of sensor A, then a service of controller B, the proxy and finally a TPA. Even if the HU database is not an active component, the resource is labeled as well so that TPA and HU Service can get access to it. The CE device is virtually labeled by the proxy after authentication, for more clarity the representations of the tag generation, granting processes as well as the TPA DLA are omitted here.

4.1.4.1 Security Evaluation of an Automotive Scenario

The DIFC does not propose any hierarchy of privileges, on contrary all services are mutually distrustful. As a consequence the effects of a successful attack or bug are limited to the tags of the labels whose applications have been compromised.

After authenticating the CE device as belonging to an authorized user, the proxy, if necessary, generates the user tags d_s and d_i and binds the device to them. If new, the proxy then explicitly grants them to the HU. The HU can after appropriately label the driver's data, label the TPA and act on the driver's behalf, e.g., by requesting a TPA DLA with d_s and d_i . CE devices are untrustworthy components, they cannot handle their own label, instead the proxy enforces at the edge of the network a DIFC-based filter. In a same way, the TPAs cannot be in charge of their label. For this purpose they run in an isolated cell and communicate through a dedicated part of the HU service

which acts similarly as a proxy between TPA and on-board services.

Controlling information flows with a TPA: The TPA is minimally trusted and is confined to the tags d_s and d_i . The secrecy tag d_s constrains the TPA to read only sensitive data belonging to the driver. The ownership of the integrity tag d_i allows the TPA to write on the driver's data. The presence of d_i in its label would force the TPA to receive d_i labeled information and prevent it from accessing other nonsensitive information like configuration files. In the considered scenario, a malicious TPA is limited to send messages only to the driver's device and can only modify his data. A label only including d_s , without d_i or ownership of d_i , would limit its data access to "read-only". Because a CE device is bound to a user's identity and communicates with the TPA, the label of the TPA is limited to tags of only one user as well, so that they can communicate together.

The isolation environment ensures that the TPA cannot exhaust or corrupt critical HU resources. The environment allows to assign a single dedicated communication interface between untrusted TPA and trusted HU service. Both system integrity and confidentiality are maintained by the dedicated HU service. On one side, it unlabels and applies DIFC filters to incoming messages. On the other side, it labels outgoing messages with a label including d_s and sometimes d_i depending on the invoked function, e. g., (d_s, d_i) for writing in the HU database.

Controlling information flows with a CE device: A CE device is restricted to the tags of its owner. It can only contact services labeled with no or with the owner's integrity tag and therefore cannot modify data considering the integrity of another user. In addition, the CE device of a user U can only receive u_s -labeled or unlabeled, i. e., nonsensitive, information, which limits the risk of information leakage. The proxy generates the new tags necessary for the devices of the new users it communicates with. It owns them and is empty labeled, in order to always be able to communicate with any new device or Internet service. The scenario demonstrates a case with a CE device, but the same DIFC mechanisms can also be extended to online services and other C2X communication partners. Either these entities are logged in on behalf of a known person, of an institution (e. g., local authorities) or directly on their own behalf (e. g., RSU).

Controlling information flows with the other on-board components: Labels can also constrain information flows between services. The HU service can receive messages from ECU A only if the HU has the tag a_s in its label or ownership. Therefore, even when forwarded by a multicast address by mistake, messages from A containing sensitive information will never be handled and sent out by the proxy or by another service without the tag a_s . On the other hand the HU owns the tags a_s and d_s , the HU can therefore make some messages from A available to the TPA through a declassification process, i. e., a careful use of the ownership to take out the secrecy tag of A from the data. An ECU B with b_i in its label, will only receive data (e. g., call to trigger a mechanism) from ECUs with b_i in their label or ownership.

4.1.4.2 Limitations and Proposed Countermeasures of the DIFC approach

A successful attack should only impact the label of the compromised services. But an attacker could also guess some tags and use forged labeled messages to get access to sensitive information or critical functionalities. In order to limit these effects, a few measures can be taken. First, the size of a service should be kept small, so that they only handle a little number of tags. Then every message should be extended with an unforgeable token specifying which tags are included in the sender's label and ownership. The token has to be signed by a trusted entity and easily verifiable by the receiver and can prevent a compromised service to use any kind of tag. These solutions should reduce the impact of an attack but were not further investigated.

TPAs are running in an environment isolated from the rest of the HU. Depending on the environment capacities, attacks like exhaustion of the cell resources can be detected and some measures can be taken. However, several users (e.g., a driver and some passengers) imply to have several environments (e.g., Virtual Machines (VMs)) running simultaneously, i.e., one per user. This approach requires a lot of resource. As a consequence, the feasibility and practicability will depend mostly on the platform capacity, e.g., memory and CPU speed, and also on the cost that car manufacturers and customers will be ready to pay.

4.1.5 Conclusion

This first approach proposes to use DIFC to solve the information security issues of the car on-board communication infrastructure. It can be easily integrated within the middleware and comply with the necessity to abstract security from the application level. Labels, ownership and DLA allow to express the security concerns of different levels of security required by all on-board applications. DIFC allows to compartment the network in labeled zones preserving the information confidentiality and car integrity. Such a model and an adapted proxy filtering are also convenient for the integration of external devices and online services. However, DIFC alone cannot mitigate the risk induced by TPAs. Carefully chosen isolation mechanisms and dedicated trusted services need to be added. In addition, since the TPAs are labeled as producing necessarily user confidential messages, the DIFC may be too rigid and unsuitable for various TPA use cases generating a large amount of non sensitive traffic but requesting as input sensitive information.

4.2 Dynamic Data Flow Tracking (DDFT)

DIFC showed to be quite suitable for static on-board use cases but less for the integration of TPAs. The DIFC approach considers indeed the TPA like a *black box*, i.e., the car is totally unaware of what happens within the executable and can only isolate it, monitor the resources it consumes, its Input/Outputs (I/Os). A solution to provide a more

fine-grained enforcement could be to get an insight of what is occurring in the running application, for example thanks to DDFT.

DDFT was successfully applied in various security domains in order to shed light on different aspects of the interactions of local and distributed components. Practically, it allows to taint and track data of interest as they propagate within a running application or system of applications. The second IFC approach of this thesis is completely independent from the first one and considers on-board services exchanging STL-based labels and not the DIFC ones. It proposes to leverage DDFT for cross-host taint tracking and to integrate it with other automotive and distributed security mechanisms, namely the SME and STL. The DDFT engine instruments the middleware of the TPA to follow and propagate STL information through the network.

After having given an overview about DDFT related work in Section 4.2.1, Section 4.2.2 describes in more details the DDFT engine and its relevancy for the automotive purpose. Then, Section 4.2.3 presents the integration of the DDFT within the automotive context and the SME. Afterwards, Section 4.3.2 provides a discussion about the approach.

4.2.1 DDFT Related Work

DDFT qualifies the action of monitoring a flow of tainted data at runtime within a running application (i. e., a process) or a running system composed of several processes. Through DDFT, *data of interest* are recognized according to predefined taint configurations and associated with metadata usually called *taint tags*.

DDFT is not a new topic, it was successfully applied for various security purposes like detection and defense against security attacks [136, 148], malware analysis [188] or privacy-oriented system monitoring [54], but also for non-security application like visualization of information flow between components of a system [131] or software debugging [8]. For the automotive context, DDFT could be applied to support the on-board security with very untrusted use cases like the integration of TPA. It would allow to track and protect privacy-sensitive information as it gets processed by the TPA and released to other on-board applications. It could also allow to install the TPA on critical ECU locations with more resources like directly on the HU and to track unsafe data, raise alerts or restrict the usage of sensitive HU operations.

Originally, taint tracking was performed to follow the propagation of tainted data in one single process, but also got extended to entire hosts thanks to VM- and emulator-based systems. The rest of this subsection provides more information on these two approaches and extends the topic to DDFT in distributed environment.

4.2.1.1 Single-process DDFT

Single-process DDFT tools [148, 98, 43, 36] instrument every machine instruction performed by a process. For this purpose, they generally make use of Dynamic Binary

Instrumentation (DBI) frameworks like Pin [114] or Valgrind [135]. They usually suffer from significant decrease of performance and need additional memory storage for taint propagation, referred as *shadow memory*. They do not require any source code modification or customized OS. Single-process DDFT was intensively investigated in order to improve its performance, e. g., TaintTrace [36] and LIFT [148] combines efficient custom instrumentation framework with code static analysis to speed-up the taint access.

4.2.1.2 Cross-process DDFT

Cross-process DDFT tools capture system-wide, generally OS-wide, data flows and mostly rely on modified runtime environments [54] or emulators like XEN [11], QEMU [13, 188, 146]. They are usually heavyweight systems requiring an extensive maintenance. They instrument every instruction performed in the host and as a consequence impose a very significant overhead for the overall system. To alleviate such performance penalties, several work tried to assist DDFT with hardware extension [45, 172]. TaintDroid [54] alleviates these issues by regarding some libraries as “trusted”, i. e., not monitored.

4.2.1.3 DDFT and Distributed Environment

Solutions for DDFT in distributed systems generally offer little reusability and require every peer to run the DDFT tool. DBTaint [46] targets data flows in SQL databases of web applications to protect their integrity, e. g., against SQL injections or XSS attacks. Neon [192] uses a modified NFS server to initialize and track the taints and to enforce adapted filtering on inbound/outbound packets. Taint-Exchange [189] proposes a generic framework based on *libdft* [98] allowing exchanges of taints over the network but without proposing any concrete security model or policy enforcement. The automotive approach [156] proposes a security model using a DDFT tool [98, 43] and network taint exchanges for every application running on the on-board network. While enhancing the security, these approaches will not meet the automotive latency requirements, if every on-board application is instrumented.

Outcome: Considering the car requirements for low latency, this second IFC approach is oriented towards efficient single-process DDFT like in [189, 156]. The TPA is DDFT-monitored, while trusted applications of the HU remain not monitored. Potential misbehavior of the TPA is locally contained by the DDFT tool. Communications between TPA and other on-board services are secured thanks to the exchange of SLT-based information and adapted policy enforcement in SMEs of trusted services.

4.2.2 Tracking and Controlling the Execution via DDFT

DDFT tools allow to monitor every machine instruction performed within a running application, i. e., to monitor every system call and to track every data flow between registers and memory. They raise a warning or stop the runtime in case of a behavior in

Source code	Shadow memory status for the tainted buffer			
	x	y	z	
0: init();	(0,0)	(0,0)	(0,0)	:0
1: buffer x = receiveBuffer(from_a_ECU);	(1,1)	(0,0)	(0,0)	:1
2: buffer y = readBuffer(from_sensitive_file);	(1,1)	(2,2)	(0,0)	:2
3: buffer z = processBuffers(x , y);	(1,1)	(2,2)	(1,1)or(2,2)	:3
4:	(1,1)	(2,2)	(1,1)or(2,2)	:4
5: write(z , in_output_file);	(1,1)	(2,2)	(1,1)or(2,2)	:5
6: sendBuffer(z , to_another ECU);	(1,1)	(2,2)	(1,1)or(2,2)	:6

Figure 4.6: Example of code with data dependencies (on the left side – in bold, the data to taint) and taint propagation (on the right side). Line 4 is empty.

contradiction with its security policies. To do so, they mostly rely on DBI frameworks, which inject custom code into the unmodified application binary depending on the invoked system call or encountered instruction, e. g., a piece of code for the enforcement of a taint-based security policies. The DDFT monitoring can be explained by looking at these three instances: i) the taint sources, ii) the taint propagation and iii) the taint sinks. The rest of this section refers to Figure 4.6 and the pseudocode it presents.

4.2.2.1 Taint Sources

Taint sources are programs or memory locations, where data enter the monitored system after invocation of a function or system call. If recognized as data of interest, they are tainted and tracked. The taint value depends obviously on the protected assets, i. e., integrity and/or confidentiality. Originally DDFT was used to protect software vulnerabilities from being exploited and a simple binary tainting was sufficient to track untrusted data, i. e., one bit of shadow memory tainting a byte of memory. But considering the goals of this thesis to both protect the system integrity and the information sensitivity, taints are required to express more values with regard to the input sources and their sensitivity level. As a consequence, the data of the TPA may be tainted depending on their STL, i. e., four bits of shadow memory tainting a byte of memory.

Then, taint sources also depends on which problem the security should tackle. The automotive scenario is less and less different from the computer world. All traditional I/O channels used by the TPA can be considered as sources : inter-process communication (e. g., pipe), filesystem, network socket and sensor input (e. g., temperature, pressure). The DDFT tool monitors the functions `receiveBuffer()` (line 1) and `readBuffer()` (line 2) and tags the buffers `x` and `y` accordingly. For instance, a buffer read from a file of the user will be tainted with a STL=(2,2) or (1,1) depending on its sensitivity, data from a file considered as the intellectual property of the car manufacturer with STL=(*,3). Data directly from a sensor input, i. e., not linked to any user identity, cannot be considered as sensitive and are tagged with STL=(0,0). The case of the network taint source is treated later in Section 4.3.1.

4.2.2.2 (intra-)Taint Propagation

All along runtime, tainted data are tracked while being copied, altered or aggregated by the application. Data resulting from different tainted inputs (e.g., `processBuffer()` line 4) receive the most relevant taint, i. e., the higher trust and security level. In the example, `z` is produced out of `x` and `y`, as a consequence the bytes computed from `x` are tainted (1,1), from `y` (2,2) and from `x` and `y` (2,2). The taints are stored and dynamically propagated in the shadow memory mapped to the real memory. For cost reasons, the number of taint value and therefore the size of the shadow memory are kept small and do not consider the data integrity. Since the development of the TPA cannot be controlled, the whole binary and especially all data it modifies or produces are considered as potentially unsafe to process. Besides considering this labeling, cases of implicit data flow are not handled by this approach [43].

4.2.2.3 Taint Sinks

Like sources, taint sinks are function calls and memory locations, where the presence of a taint is checked in order to enforce a policy. The policies concern decisions about transmitting data to a specific function, or using the data as program control data (e.g., return address). Like sources, they are problem-specific. In the current case it concerns functions and system calls writing to a standard output (e.g., in a file, `write()` on line 4) or sent them over the network (`sendBuffer()` on line 5). For writing in a file, the data and file sensitivity have to be similar, i. e., private data in user's personal files and sensitive data of car manufacturers in files labeled accordingly. For this purpose, the DDFT engine lists all accessible files and maps them with the data sensitivity that they contain and can receive. As mentioned earlier, the case of the network taint source is treated later in Subsection 4.3.1.

4.2.3 Middleware-based propagation of DDFT taints

If well configured, DDFT tools allow locally to eliminate numerous attacks related to stack pointer overwriting, like buffer-overflows [4], format-string exploits [158] or ROPs [160] while following the propagation of sensitive information. However, other trusted automotive services cannot benefit from such security mechanisms without dramatic performance penalty. As a consequence trusted services rely on the secure implementation of their middleware and their SME, to protect them against attacks from the TPA. The IFC between service and TPA is ensured by DDFT tool via exchange of STL metadata through the in-band middleware protocol.

4.2.3.1 (extra-)Taint Propagation

Figure 4.7 graphically presents a few taint sources and sinks as well as the propagation of taints between a TPA and a service present on other ECUs. The taint propagation

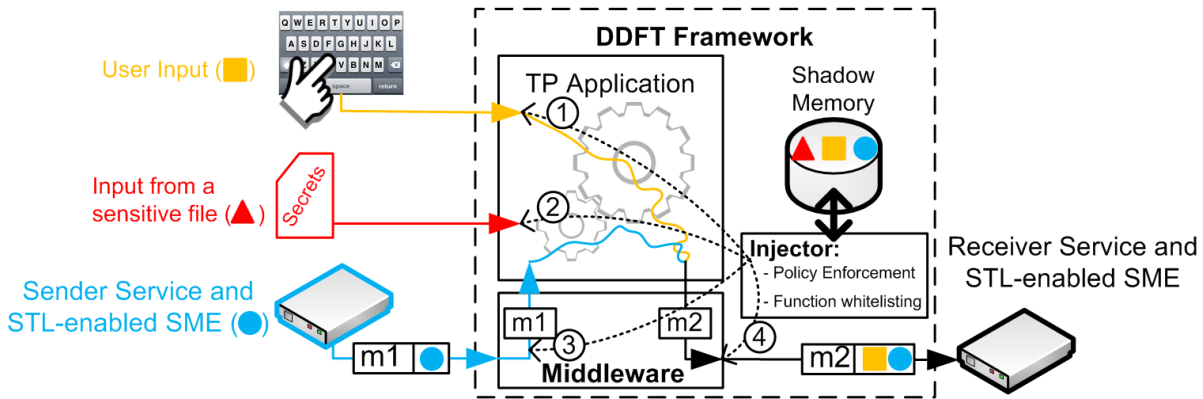


Figure 4.7: Overview of the DDFT framework in the on-board network. The solid lines show the I/O data of the TPA. The colored shapes represent different levels of sensitivity that are expressed by the taint values (i.e., yellow square (1), red triangle (2), blue round (3)). These STL taints are injected using binary instrumentation (*Injector*). The *Injector* monitors the execution, especially system calls (dotted lines) and the taint propagation between memory and registers. $m1$ and $m2$ are tainted messages sent respectively to and from the TPA. The TPA output $m2$ shows a combination of sources “round” and “square” but not “triangle” and is therefore tainted accordingly.

mechanisms between the TPA and a HU service are performed through the middleware and follow the same principle. The system calls, related to the network socket management (lines 2, 6 of Figure 4.6 and bullets 3, 4 in Figure 4.7) are intercepted by the *Injector* of the DDFT tool. For inbound messages (bullet 3), the *Injector* checks whether the emitting service is allowed to communicate with the TPA, extracts the STL_{REQ} of the payload from the middleware header and taints the received data in its shadow memory. In the case of data received from the proxy, where no STL_{REQ} but a STL_{STATUS} is specified, data are tainted as being private, e.g. with $STL=(1,1)$ or $(2,2)$ depending on the data source. For outbound messages (bullet 4), the *Injector* checks if the TPA is allowed to communicate with the addressee and adds the STL taints related to the message payload in the middleware header. Both sides of the communication establish a secure communication channel. After the message reception, the middleware of the receiving side extracts the taint value from the payload and enforces the suitable security policies.

4.2.3.2 STL-based Enforcement

Unlike the DDFT tool, the middleware and the SME of an on-board service does not show much flexibility. They enforce static policies and cannot be aware of the requirements and specificities of each new TPA. The middleware therefore enforces a taint-based filtering involving generic rules for all TPAs. On one hand, the monitored TPA establishes

a communication channel over a dedicated interface of the middleware to make the SME aware of the communication with an untrusted TPA. On the other hand, the middleware trusts the DDFT tool to provide accurate taint values. Based on this information the SME decides, like presented in Chapter 3, whether the service can safely process the data and whether the service is allowed to receive data with such a sensitivity.

4.2.3.3 DDFT Configuration and Security Policies

TPAs or their middleware cannot be trusted to enforce any security policy, instead all policies are managed and enforced by the DDFT tool. Both static and dynamic rules can be distinguished:

a) Static rules: These rules are embedded in the DDFT tool. They mostly concern the STL taint management, i. e., taint values, definition of taint sources, propagation and sink rules. They are defined by car manufacturers and cannot be overridden by the dynamic rules. Additional rules constraining the behavior of a TPA can be applied to prevent basic DoS attacks, e. g., the DDFT engine can monitor the TPA outputs to limit their size and their emission rate

b) Dynamic rules: These rules are loaded with the TPA in a rule set, similar to the one provided by an Android application. They define which on-board and C2X communications are authorized and specify the trust level of online services and devices, they can communicate with. This rule set has to be approved and signed by the car manufacture after a testing process. Moreover, a TPA may ask the DDFT tool to declassify some data, i. e., to taint them with a lower STL in order to send them to a less trusted service. These cases have to be specified in the rule set as well and concern the driver's data only. For example the declassification of private information may trigger the display of a warning pop-up asking for the driver's approval.

4.2.4 Discussion

This subsection refers to the attack scenarios presented in Chapter 2 and describes how this second IFC approach would react under attack. Both scenarios feature an attacker getting control of the TPA by launching for example an attack related to the overwriting of a stack pointer. By design, the DDFT can detect such exploits and stop the program. As result, an attacker cannot compromise the TPA integrity to perform the any of the 2 scenarios and has to leverage another system weakness.

4.2.4.1 About the Integrity Attack Scenario

This scenario considers an unauthorized access of a HU resource (e. g., file or process) aiming at disturbing the platform functioning. A DDFT engine is traditionally used to track information flows in the application memory but they also monitors every invoked system call and function. It can therefore blacklist the functions and processes that the

TPA should not get access to and can restrict its file access in writing and reading. This scenario also considers the case of a TPA sending bogus packets to an ECU in order to disturb its functioning. In a similar manner, the DDFT engine controls the socket management and only allows communication with authorized ECUs. Then based on the received STL, provided by the DDFT engine, the ECU is aware of the potential risk and adapts its packet processing.

4.2.4.2 About the Confidentiality Attack Scenario

This scenario mostly considers the release of sensitive information to the outside. TPAs receive information through multiple ways: by accessing shared memory, via filesystem access, with inter-process communications or from the network. On one hand, the DDFT engine monitors every of these input channels and taints data coming from them according to their sensitivity. On the other hand, the only way to release the information is via the on-board network and then over the proxy. This work does not consider information leakage through other means, i.e., physical port of the ECU (USB), or display screen. The DDFT tool monitors the socket, which tainted information is going through and to which destination and can therefore block an unauthorized flow. If the DDFT tool cannot enforce a decision, the addition of the STL_{REQ} value in the message header allows the proxy to enforce a final decision based on the actual information sensitivity.

4.2.4.3 About the Approach

Unlike OSs like Android, which control applications with a limited set of coarse permissions, this IFC approach allows a very fine-grained security enforcement. The DDFT tool monitors every invoked function, every I/O channel of the TPA and tracks every byte of the application memory. The taint values, coded over four bits, offer sixteen different values expressing as much sensitivity levels. Such monitoring allows the TPA to remain functional even when simultaneously handling very sensitive data and communicating with untrusted sources. Considering the example shown earlier in Figure 4.7, a TPA takes as input non-sensitive ($TL=0$) and sensitive ($TL=3$) data, but is still able to generate outputs tainted as “*not containing any sensitive information*” ($TL=0$) and send them out. Monitoring the middleware and injecting taints allows to export the local DDFT benefits over the network. The in-band middleware protocol makes on-board applications information security aware and contributes to a homogeneous security enforcement in the whole car.

4.2.5 Conclusion

This approach makes use of the complete architecture proposed in Chapter 3, i.e., without modification of the in-band middleware protocol, as the first approach is suggesting

to. It extends the security architecture with a DDFT-based monitoring of the TPA and adequate TPA–service communication interfaces. In comparison to the approach taken in Section 4.1, DDFT allows to consider the TPA as a “grey-box”, i. e., without previous knowledge of the TPA, to partly understand and control what is performed within the untrusted application. DDFT intercepts I/O related system calls and propagate fine-grained tainting information all along the TPA data processing, i. e., from the data entrance until their release to the network. Via a customizable API, the DDFT tool provides a flexible definition of taint sources, sinks and taint propagation policies. DDFT operates transparently in unmodified OSs, independently of the hardware platform and allows real-world legacy applications and TPA/middleware to be efficiently monitored.

4.3 Combining DIFC/DDFT

While providing locally on the HU a very fine-grained control over the TPA, the STL/DDFT approach lacks a formal security model for ECU-to-ECU communications. On the other side, the DIFC-based approach suffers from its rigidity and lack of control over the TPA. As a result the third approach proposes to combine both of them and to leverage a formal authorization model for communications between on-board services via DIFC and the DDFT engine to control the TPA. Practically, depending on whether they are authorized to communicate with a TPA, the middleware of a service and its SME may dispose of two communication interfaces. The first one is designed for a DIFC-based serialization of the middleware header and the second one optimized for headers containing a STL taint. Figure 4.8 pictures an Ethernet/IP-based network including three ECUs and describes how the security mechanisms are enforced. HU and controllers may communicate with the TPA and are therefore DIFC/DDFT-enabled, while sensors may not and are only DIFC enabled. Applications allowed to communicate with the TPA present two communication interfaces embedded within the middleware. The first one is dedicated to TPA communications and enforces the STL approach. The second one is set for regular on-board traffic and enforces the DIFC policies.

The rest of the section is organized as follows. The architecture specifications of the DIFC/DDFT coupling and related interface policies are presented in Section 4.3.1. Then this last IFC approach is briefly discussed and compared with the 2 others in Section 4.3.2. Finally Section 4.3.3 proposes a security conclusion for this chapter.

4.3.1 DIFC/DDFT-enabled Middleware

The DIFC/DDFT interfaces concerns the middleware layer of services establishing direct communications with the TPA. It allows them to interpret STL taints based on their DIFC label. Like for the DIFC- or STL-based approach, the applications of a service are unaware of this interface and its enforcement.

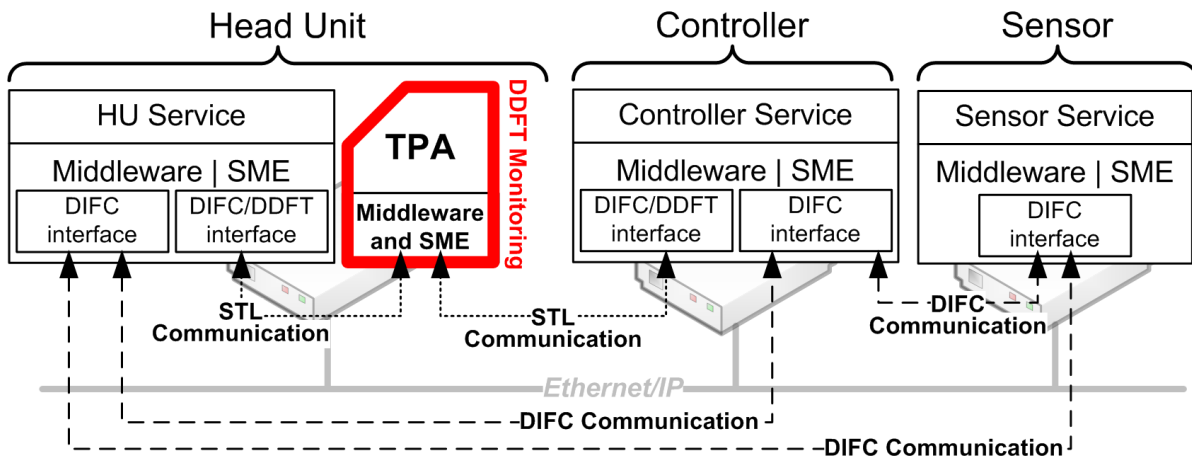


Figure 4.8: Architecture for DIFC/DDFT coupling. This on-board network features three ECUs: a sensor, a controller and a HU. A TPA is running on the HU and monitored via DDFT. Only few applications of the HU and of the controller are allowed to communicate with the TPA via a customised middleware offering two interfaces. The first one is dedicated to TPA communications (arrows with short dashes) and follows the STL approach. The a second one is for regular on-board traffic (arrows with long dashes) and follows the DIFC approach.

4.3.1.1 Design Choices

Firstly, the TPA and the DDFT engine are not part of the DIFC model and are not assigned any label. As a result, the TPA is allowed to receive information from the whole car. At the same time, the DDFT engine provides accurate STL taints to communicating remote services and gives them a precise idea of the sensitivity of the TPA output. For example, like in Figure 4.7, even if the TPA receives as inputs highly sensitive data, the output STL may still indicate that the payload was processed out of non-sensitive data. Contrary to the DIFC approach but similarly to the STL/DDFT one, this third approach allows the TPA to remain very functional, even when handling highly-sensitive data.

Secondly, in this approach, TPAs and DDFT engine only receive messages with STL taints. This allows to keep the DDFT engine simple, efficient and generic and not to worry about the different architecture specifications of all car models (e. g., different tag identifiers). Then, TPAs are most likely to receive information from on-board services and from the outside, process them and communicate towards the outside. The amount of traffic from the TPA to on-board services will remain minor. As a consequence, the security should be based on a taxonomy oriented towards a secure information release with the outside, thus towards the STL.

Thirdly, due to a limited number of taints and a high risk for privacy, the data

of only one user should reach the TPA. As a consequence, all monitored TPAs are assigned one Car User IDentity (CUID). Like for the user tags, these CUIDs are defined and distributed by the proxy. For each message exchanged between a service and a TPA, a STL and a CUID are added in the header. The DDFT tool filters inbound messages based on the provided CUID. The middleware of the concerned service can easily characterize whose privacy is concerned and ensure the transition between the STL and DIFC models.

Lastly, like the DIFC approach, the DIFC/DDFT approach does not make use of the `STLSTATUS` or `-REQ` for communications between on-board services. The addition of such a field in the middleware header could obviously improve the reactivity of the system to dynamic use cases. But cars include very static architectures and C2X communication channels involving on-board services, which are predefined and setup in a secure way. The addition of a STL would as a consequence be superfluous.

4.3.1.2 DIFC/DDFT-enabled Service

As a consequence, all on-board services can send data to the TPA. Their middleware just provide the right CUID (if these data are private) and a suitable STL value. The DIFC labels are enforced to create information flows respecting their integrity and confidentiality. Since the integrity of the TPA outputs cannot be assessed, the transition between DIFC label and STL taint only focuses on the information confidentiality.

Traffic “On-board service → TPA”: For messages flowing from an on-board service to a TPA, the sent STL value depends on the secrecy labels of the service:

- for a label involving tags expressing a high secrecy for the car manufacturer, the STL gets a TL=3. The SL is not relevant since the data will not leave the car. The list of high-secrecy tags is defined by the car manufacturer and available in each service interface.
- for a label involving tags expressing the secrecy of a user, but not expressing a high secrecy for the car manufacturer, the STL gets a TL=2 or 1 depending on their privacy level. The SL depends on the user’s settings since it is her own data, but as a default value a SL=2 is advised.
- in any other case, the STL gets a TL=0 and a SL=0.

Traffic “TPA → On-board service”: For the opposite case, i.e., when the service receives a message from the TPA, the service middleware decides whether to pass the data to its applications based on the received STL and its own label (a star in the STL description may characterize any kind of value between 0 and 3):

- a STL=(*,3)¹ forces the middleware to pass the data to an application handling highly confidential data that cannot leave the car. Thus an application having a user secrecy

¹STL=(SL,TL)

tag in its label is not able to receive such data. An authorized service should also include high secrecy tags of the car manufacturer in its label.

- a STL=(*,1) or (*,2) forces the middleware to pass the data to an application handling the private data of a user. Therefore a service with a user's secrecy tag corresponding to the received CUID field should be able to get the data.
- a STL=(*,0) indicates that the data are not sensitive and can be passed to all kind of applications.

These last rules do not consider the SL part of the STL. If sent to the outside, a communication from a service has to go through a DIFC-based enforcement at the proxy level, which is already statically configured. The SL is more relevant for unknown and dynamic cases where security has to be evaluated and configured on-the-fly, like for C2X communications involving a TPA.

4.3.1.3 DIFC/DDFT-enabled Proxy

Most TPA outputs will aim toward the outside. As a consequence, in addition to the traffic regulated by the DIFC model, the proxy will face STL-based traffic from the TPA and will have to enforce a suitable filtering. For the TPA traffic, the proxy enforces more than just a STL-based filtering. The proxy also makes sure that the CUID is appropriate for the communication, e.g., a received CUID of a user U communicating with U's CE device or an online service logged in with U's personal account. Thus, the proxy provides two types of filtering: (1) STL-based for communications with the TPA and (2) DIFC-based for communications with other on-board services.

4.3.2 Discussion

This section proposes first to briefly discuss the DIFC/DDFT approach. In a second phase, a comparison table as well as relevant evaluation criteria are given in order to pinpoint the strengths and weaknesses of the three approaches presented in this chapter.

4.3.2.1 About the DIFC/DDFT-based approach

On one hand, the DIFC/DDFT approach leverages the DIFC model for all ECU-to-ECU communications and on the other side a DDFT monitoring and STL labeling techniques for TPAs. This last approach does not define a new IFC concept, but rather shows the feasibility of coupling two different types of IFC methods. At a practical level, it benefits from the formalization of the DIFC model for static use cases and of the flexibility of the DDFT engine for use case requiring a more dynamic security enforcement.

4.3.2.2 Comparison of 3 IFC approaches

Table 4.1 proposes to summarize and compare the three IFC approaches. The evaluation criteria, used to establish this table are based on the attack scenarios presented in Chapter 2 and are formulated as yes-no questions. Depending on the answer more technical details can be provided. The answer “Yes/No” expresses a partial fulfillment of the question. The questions are listed here as follows:

- **“Enforcement” Criterion:** This criterion specifies where and how the security is enforced, whether the security takes data confidentiality and integrity into consideration and what kind of in-band protocol is used.
- **“TPA Monitoring Paradigm” Criterion:** This criterion specifies which monitoring or “box” approach is applied to the TPA, i. e., *black* for an I/O monitoring or *grey* for an I/O monitoring with introspection of the TPA.
- **Integrity Criterion 1 (IC.1)** - Does the approach prevent ECUs from being attacked/compromised by an attacker, i. e., having physical access to the on-board network or access through the proxy interface?
- **Integrity Criterion 2 (IC.2)** - Does the approach prevent the TPA from being compromised by an external attacker? i. e., may an attack be detected before it actually compromise the TPA?
- **Integrity Criterion 3 (IC.3)** - Does the approach prevent HU resources from being exhausted by a malicious TPA and as a consequence from disturbing the standard HU functioning?
- **Integrity Criterion 4 (IC.4)** - Does the approach prevent remote ECUs from being attacked/compromised by a malicious TPA?
- **Confidentiality Criterion 1 (CC.1)** - Does the approach prevent the TPA from stealthily stealing sensitive data from user or the car manufacturer, e. g., via direct file access? The category *information access* describes how the access to sensitive information is restricted.
- **Confidentiality Criterion 2 (CC.2)** - Does the approach provide a privacy-aware information release with external C2X communication partners? The categories *Granularity* and *Flexibility* designates respectively at which granularity the access control can be performed and for which types of use case it is the most suitable.
- **Confidentiality Criterion 3 (CC.3)** - Does the approach protect on-board applications from a system weakness leveraged by an external attacker, e. g., leveraging an application or network routing misconfiguration? The categories “*TPA*

Table 4.1: Table comparing the three IFC approaches of Chapter 4.

Evaluation criteria	IFC approaches		
	Pure DIFC based	STL/DDFT-based	DIFC/DDFT-based
Enforcement		Distributed and Middleware-based	
• for integrity	Yes, all traffic	Yes/No, incoming C2X traffic	Yes, all traffic
• for confidentiality	Yes, all traffic	Yes, all traffic	Yes, all traffic
• in-band protocol	DIFC labels	STL labels	DIFC & STL labels
TPA Monitoring Paradigm	black-box approach	grey-box approach	
Integrity			
• IC.1	Yes/No, integrity protection support based on DIFC information	Yes/No, integrity protection support based on STL _{STATUS} information	Yes/No, integrity protection support based on STL and DIFC information
• IC.2	No, but TPA isolation	Yes	Yes
• IC.3	Yes, by TPA isolation and I/O filtering	Yes, via binary instrumentation and I/O filtering	Yes, via binary instrumentation and I/O filtering
• IC.4	Yes/No, labeling of the TPA I/O with integrity labels	Yes, TPA output monitoring and communications via dedicated channels	
Confidentiality			
• CC.1	Yes, isolation of the TPA	Yes, instrumentation of the TPA	
<i>information access</i>	Via static dedicated services	Whitelisting of authorized API functions and files	
• CC.2	Yes, Proxy filtering based on the set of user tags and for multiple users	Yes, Proxy filtering based on the STL taxonomy, but for a unique user	Yes, Proxy filtering hybrid, based on tag and STL for multiple users
<i>Granularity</i>	suitable for static use cases	suitable for dynamic & static use cases	
<i>Flexibility</i>			
• CC.3	Yes, filtering and labeling of TPA I/O	Yes, TPA memory and I/O tainting	
<i>TPA is targeted</i>	Yes, DIFC labeling of on-board communications	Yes, STL labeling of on-board communications	Yes, hybrid traffic labeling (DIFC- and STL-based)
<i>A service is targeted</i>			

is targeted” and “*A service is targeted*” specify which IFC mechanisms are used depending on the attack target.

DIFC/DDFT combines all advantages of the DIFC and the DDFT approaches and can be considered as the best solution from a pure security point of view. However, as indicated in the related work of the two first approaches, such security mechanisms may suffer from significant security penalties. These technical issues need and will be investigated in Chapter 5.

Additionally, regarding the TPA monitoring, it seems obvious that the monitoring solution has to be defined based on the TPA and its technical characteristics. A TPA that can be installed that can be directly installed on the HU may be monitored by DDFT or by DIFC in an isolated environment, whereas a TPA requiring a specific OS like Android will be monitored by DIFC and installed in a separated partition of the HU.

4.3.2.3 Security Gains:

The beginning of this section compared three IFC approaches based on general criteria. The rest of this section proposes to evaluate the same approaches with concrete attacks performed on the use case of Section 2.3.3. Attacks are separated between 1) “attacks” trying to bypass or not respecting the access control of the car (e. g., malicious but valid software of a subcontractor running on the middleware of the car manufacturer) and 2) the TPA-based ones. First a short description of the attacks is provided, then the responses to these attacks are summarized in Table 4.2 and 4.3.

Attack 1.1 - Unauthorized function triggering through intermediary ECU: B triggers a function on C on behalf of A, whereas A is not authorized to trigger any function of C. The letters may be replaced by an automotive middleware. For example, ABS tries to trigger the “Sport” mode of the PCM via the HU. The HU has a function interface for that, but the ABS is not authorized to communicate with the PCM.

Attack 1.2 - Unauthorized information access through intermediary ECU: B retrieves information from C, initially requested by A and forwards them to A. But A is not authorized to communicate with C. The letters may be replaced by an automotive middleware. For example, the TPMS, requests from the ABS information from the HU about the car status, e. g., maintenance information, to send them to the original software subcontractor via radio frequency. The TPMS is not authorized to communicate with the HU.

Attack 2.1 - Unauthorized function access from a TPA: Unlike the subcontractor software, where the middleware can be trusted because it was developed by the car manufacturer. The middleware of the TPA should not be trusted. A TPA could therefore leverage the middleware protocol (and for example the IFC metadata) and trigger a function it should not. For example, a TPA may want to disable the ABS, while using the secure communication channels opened between HU and ABS.

Attack 2.2 - Unauthorized data release from the TPA: Similarly, a TPA could leverage the middleware protocol (and for example the IFC metadata) in order to retrieve sensitive data and send them to an untrusted online server. A TPA may be able to get information from the GPS, ABS and HU, aggregate them and forward them outside. In order to work the TPA needs all these information but should anonymized them, in this case the malicious TPA will try to send them with a CUID.

Attack 2.3 - TPA DoS attack: the TPA either 1) excessively consumes the HU resources, 2) sends messages over the network with an excessive throughput rate, or 3) sends a remote attack to another ECU, e. g., buffer overflow.

Additional notes: Simple cases of an ECU trying to trigger an unauthorized function or to get unauthorized data was previously discussed in Section 4.1.4.

The dedicated TPA service mentioned in Table 4.3 is a middleware service between the isolated TPA and the communicating middleware. All traffic to and from the TPA goes through it. This service is besides responsible to enforce DIFC rules instead of the TPA. More details about this dedicated TPA service are provided in Section 5.4.1.

This discussion does not consider the case where an ECU gets compromised. The whole IFC framework relies on the middleware of all ECU being trusted. Measures to cope with compromised middleware were discussed in Section 4.1.4.2.

4.3.3 Conclusion

The DIFC/DDFT approach results in the combination of two IFC approaches, also presented in this chapter. First, a DIFC-based model is enforced on on-board communications between services and provides formalism for the establishment of integrity and confidentiality zones (expressed by the tags in the service labels). Regarding the TPA integration, a major issue of the DIFC solution was its staticity and lack of flexibility as for I/O monitoring. Unlike the isolation cell, the DDFT approach provides a flexible framework for a secure TPA integration. While allowing a deep monitoring and complete control over the TPA, the DDFT can be easily coupled to the DIFC model via DIFC/DDFT interfaces. From a security point of view, this approach seems to be the most suitable to simultaneously secure on-board communications, integrate TPAs and C2X communication partners.

4.4 Summary

Automotive IFC is about monitoring and controlling how information is propagating over the on-board network and within the ECUs. Consistent with the Chapter 3, this chapter describes the integration of IFC techniques at the middleware level. Enforcing information security at this layer allows to abstract the security and especially the need for confidentiality (e. g., privacy, corporate secret) and integrity in security zones, where

Table 4.2: Table comparing the three IFC approaches for attacks on the access control. A \checkmark at the end of the attack response means the attack is defeated. A \approx means that the attack is mitigated. A \times means that the attack is not defeated.

	Pure DIFC based	STL/DDFT-based	DIFC/DDFT-based
Attack 1.1:	The ABS is labeled in integrity with abs_i . The function of the PCM triggering the sport mode on the PCM is labeled pcm_i . The only way of triggering this function via the HU is to have pcm_i in the HU label. But it is not the case. Plenty of data handled by the HU do not present the requested integrity level. The HU owns pcm_i and can add it to the message label if a policy allows it, e.g., function triggered by the driver from the touch screen of the HU. Since the call is received from the ABS, no policy allows it, the HU forwards the call without pcm_i , the call is therefore dropped by the PCM. $\rightarrow \checkmark$	The STL does not consider integrity, so if the forwarding interface on the HU and the secure communication channel to the PCM exist, the call is forwarded. $\rightarrow \times$	Same answer as in the case Pure DIFC based. $\rightarrow \checkmark$
Attack 1.2:	The TPMS is labeled in secrecy with $tpms_s$. The function of HU to retrieve maintenance information is labeled with hus . The only way of getting this information is to take out $tpms_s$ from the message label and add hus in the new message label. The ABS owns hus but like in the previous case has to evaluate if there is a policy to authorize the forward of this call. There is no policy, so the call is forwarded with just $tpms_s$ and dropped by the HU. $\rightarrow \checkmark$	The TPMS is forwarded with a STL=(0;0), since it can send information in plaintext via radio frequency. Therefore all received sensitive information, e.g., with STL=(1,1) like the maintenance status will be dropped by the middleware of the TPMS itself before the information reaches the application level. $\rightarrow \checkmark$	Same answer as in the case Pure DIFC based. $\rightarrow \checkmark$

Table 4.3: Table comparing the three IFC approaches for TPA-based attacks. A ✓ at the end of the attack response means the attack is defeated. A ≈ means that the attack is mitigated. A × means that the attack is not defeated.

	Pure DIFC based	STL/DDFT-based	DIFC/DDFT-based
Attack 2.1:	Considering the safety criticality of the function, the dedicated service has no static interface to forward this call. → ✓	The ABS IP address can be blacklisted by the DDFT tool. If the TPA call this address, the DDFT tool stops the TPA. → ✓	Same answer as in the case STL/DDFT based. → ✓
Attack 2.2:	When receiving private data, the TPA is labeled per DLA as sending private information. The proxy considers all labeled messages from the TPA as private and never forwards them to an untrusted server, even if they do not include private data. → ≈	By tracking in the TPA memory, whether the sent buffer includes private information, the DDFT accurately taints the message by injecting the taint just before sending the message. The proxy thus decides, depending on the taint, if it can forward it. → ✓	Same answer as in the case STL/DDFT based. → ✓
Attack 2.3:	1) TPA is isolated in its cell → ✓ 2) the dedicated TPA service filters based on the throughput frequency → ✓ 3) the middleware of other ECUs is supposed to be verified and protected against these attacks. → ≈	1) DDFT does not provide a way to limit the resource consumption (CPU or memory) of the TPA. The TPA should be tested on before hand. → × 2) DDFT can detect the emission throughput and close the TPA if it goes over a pre-defined limit. → ✓ 3) Same answer as in the case Pure DIFC based → ≈	1) Same answer as in the case STL/DDFT based. → × 2) Same answer as in the case STL/DDFT based. → ✓ 3) Same answer as in the case Pure DIFC based. → ≈

the information can freely be exchanged. The middleware makes sure that these information do not trespass the zones they have been assigned to, or if they have to, it makes sure that they follow the IFC policies defined by car manufacturers and car users.

The first IFC approach consists in a custom DIFC model. DIFC can be efficiently adapted to distributed systems, where an application or a group of applications can be in charge of their resource management. For the automotive scenario, the middleware and its applications are labeled together depending on their confidentiality and integrity requirements. Different rules, called partial order, and communication labeling techniques can be applied in order to exchange security metadata and control the data flow of each on-board communications. It can also be applied to secure the integration of external communication partner and TPA by means of isolation mechanisms and DIFC filters enforced in dedicated communication proxy and on-board services.

While providing a formal model for information security, this approach remains too static and rigid for unconventional use cases. A second and more flexible approach combines STL-based enforcement and DDFT monitoring techniques. The DDFT engine allows to fully control the TPA and the information propagation, while the STL proposes a very flexible and simple security model for C2X communications.

The third approach leverages the two previous solutions: (1) DIFC model for static on-board communications and a STL-enabled DDFT engine for TPA monitoring. The coupling between the two models is performed by dedicated DIFC/DDFT interfaces allowing simple correlation rules between DIFC labels and STL taints. This third solution seems to be optimal in order to provide security with enough flexibility and adaptability. However DIFC and DDFT mechanisms are often subject to significant performance penalties and a rigorous validation of these approaches would not be relevant without a proper functional evaluation. In the next chapter, the implementation specifications of these approaches are described and some performance benchmarks are performed in order to estimate the relevancy of these solutions.

Prototypical Evaluation and Discussion

While developing several security approaches for middleware-based security and Information Flow Control (IFC), previous chapters addressed the implementation questions only superficially. Automotive requirements are very demanding and the security concepts of this thesis involve several distinct components that should be combined and tested together. A proof of concept as well as a first functional evaluation are required to prove the relevancy of this overall security architecture. Besides, Ethernet/IP is just coming in the car development process, which means that no real-world platforms are available for now. As a consequence a testing environment fitting the future hardware specifications of the car needs also to be defined.

This Chapter aims at providing concrete information about the implementation of security mechanisms at the middleware level and about the first evaluation phase. The security architecture has been implemented and integrated within an IP-based middleware, which is currently considered for an automotive purpose.

Firstly, the evaluation methodology as well as some benchmarks are presented in Section 5.1. Then the implementation specifications of each major component are described, tested and evaluated. Section 5.2 provides an evaluation of the security-enabled version of the *Etch* middleware [56]. Section 5.3 deals with its associated communication proxy and Section 5.4 with two Third-Party Application (TPA) monitoring types, namely Decentralized Information Flow Control (DIFC) plus isolation based on the XEN[®] hypervisor [11] and DIFC plus custom version of the Dynamic Data Flow Tracking (DDFT) engine *libdft* [98]. Finally all benchmark results are discussed together with the previous security discussions in Section 5.5, and provide a first validation of these concepts.

5.1 Evaluation Methodology

The main goal of this section is to assess whether the middleware and its security can be evaluated in a suitable way and if so, in which manner. By definition, the middleware is a software layer providing flexible technical capabilities in term of networked services,

distributed resource management and security. This layer may serve as basis to build and run new applications. As a consequence the middleware and its security have to be well-architected, reusable and provide a practical development process.

Traditionally, software and other IT systems are evaluated based on classical criteria like performance, usability, scalability, security and robustness [68]. But the middleware is particular, instead of filling a particular role, it can be seen as the Swiss army knife of the software layer. Two challenges, called the *Middleware-Design & Evaluation Gaps* can be identified [51] and be extended to a security middleware.

- **Middleware-Design Gap:** The car middleware needs to anticipate all requirements of future on-board automotive applications. Without becoming too complex, difficult to understand, use and maintain, this software has to be complete and sufficient for all expected use cases. Chapter 2 presented the IP-based automotive use cases and the following chapters presented how and where security in the middleware should be applied.
- **Middleware-Evaluation Gap:** The second challenge is about choosing what to assess, i. e., which parts/characteristics of the middleware and which benchmarks to consider. For a complex system architecture like cars, the middleware is used for a plethora of use cases involving additional and non-middleware-related technologies. Several questions arise as for the evaluation: Which test applications have to be considered? Which user features (e. g., for the developer) can be characterized as appropriate? Which results obtained with a particular application X can be imputed to the middleware? Are standard evaluation methods still valid? Is it possible to evaluate a middleware out of the context of an application?

The remaining of this section provides answers for an automotive security context. Subsection 5.1.1 describes the process of evaluating a middleware security proof-of-concept. This subsection also presents which types of application and tests have been performed. Subsection 5.1.2 presents the environment, in which such a new automotive technology can be tested. Subsection 5.1.3 provides information and criteria about automotive software development.

5.1.1 Functional Evaluation of a Secure Runtime

Such an evaluation basically concerns how the security middleware reacts and performs during runtime. The design and setup phase are out of scope for now but will be considered in the next sections. The core use cases of an automotive middleware are an efficient RPC and notification capacities as well as opening secure communication channels providing authentication, encryption and access control.

A danger, when implementing new software components, is to get lost in an explosion of features. The first motivation when designing an automotive security middleware

should be to keep it simple. A first lightweight version, in C code and focused on the core functionality of the middleware, was developed and extensively tested. A second more complex Java version was extended and tested for more ambitious use cases within the car. Both versions are later described in Section 5.2.

Evaluation tests should exclusively stress the middleware and its security mechanisms. The complexity of the application running on top of it is therefore kept at its minimal in order to focus the measurements on the performance of the secure middleware.

Scalability and robustness can be judged as less relevant for a proof-of-concept and are out of this scope. Considering that the car infrastructure consists of 80 ECUs and roughly the same number of middleware instances, the scalability need is relatively limited. Besides the robustness criteria and tests for automotive software are already defined in the ASIL standards [94].

Performance measurements allow to quantify the overhead of a system consisting of several entities (i. e., clients and servers) communicating together over C2X or on-board communication channels. For this purpose, measurements are repeated for different situations (e. g., on-board client and server, on-board server and a client CE device) and for different types of traffic (i. e., exchanges of large objects or short notifications). Both bandwidth and throughput should be computed. The bandwidth is a good performance indicator to characterize the potential of exchanging large objects, whereas the throughput is better to determine the system suitability for short-message communications at high frequency.

Sections 5.2, 5.3 and 5.4, demonstrate the communication overhead for respectively middleware (i. e., the basic components), proxy and TPA monitoring features. In order to put in perspective the performance costs induced by each security mechanism, measurements fall in several categories:

- a) *Blank measurements.* They are performed without enforcing any kind of security (i. e., plaintext communications) and give an offset value in order to evaluate the security impact.
- b) *Minimum security measurements.* They are resulting from entities communicating over secure communication channels, i. e., providing authentication, integrity and encryption through secure communication protocols like IPsec or SSL/TLS. They provide a lower bound overhead imposed by a minimal security setup.
- c) *DIFC-based measurements.* They highlight the performance penalties induced by the enforcement of DIFC-based policies on on-board communications between trusted services.
- d) *TPA monitoring measurements.* They show the performance impact caused by a secure TPA monitoring, i. e., either by DIFC & isolation or by DIFC/DDFT-based framework.

For each measurement result, 10 measurements of 10.000 calls-plus-responses were performed in order to calculate their means. The measurement scenarios are specified in more details later in this chapter.

5.1.2 Testing Environment

As motivated in Chapter 2, Ethernet is already available in luxury cars for the diagnosis purpose [157]. However the functional integration of Ethernet/IP for car has only been recently investigated and promoted by the SEIS project [63]. Its introduction into the car is only planned for 2018 [157]. While the future Ethernet/IP-enabled platforms and most of their technical specifications still remain unknown, certain indications lead to think that car manufacturers will equip their car with platforms more powerful than today [21, 175]. On one side ambitious use cases for automated driving and infotainment will require more resources. But on the other side a trend for simplifying of the on-board E/E architecture emerged and tries to reduce the number of ECUs by means of capable paravirtualized and/or multicore platforms [134, 124].

In order to test Ethernet/IP for cars, two options are available: 1) installing an Ethernet/IP network coupled to a CAN bus in the car trunk or 2) setting up on a table an on-board network with ECUs or commodity platforms. The Java-based implementation has been tested for both approaches, the C version only with the second one. The results of the next sections are performed with the second testing approach. The considered testbed consists of computers interlinked with Gigabit Ethernet and running standard 32-bit Fedora Linux on an Intel Atom N270 (1,6 GHz) with 1GB RAM. This network is composed of two or three computers representing: a ECU server and a ECU client or a CE device, a proxy and a HU. The configuration of these platforms is comparable to current UNIX-based HUs, which operate at 1,3 GHz [20]. The experiments have been conducted when the hosts were idle, i. e., with no other user processes running simultaneously.

5.1.3 Engineering-driven Middleware Development and Setup

It concerns the evaluation of factors related to the different phases of the car conception: *Development*, *Production* and finally *Post-Production* phases. These phases are subject to numerous functional requirements. The design, implementation and setup of the security should be accordingly adapted.

5.1.3.1 Development Phase

All along this phase, which lasts approximately 4 to 5 years, car manufacturers design/refine the car E/E architecture. It includes choosing all expected functionalities, communication interfaces and security mechanisms. Based on detailed bills of specifications, car manufacturers and subcontractors produce and test successive versions of

ECU platforms and/or software. After what, the ECUs are cabled together, the software components are configured (e. g., settings, policies, key distribution) and the last set of tests is performed on a complete test car.

Development Evaluation Criterion: The software development is an iterative process, the software goes back-and-forth between testing and refinement. Car manufacturers, who have a full overview of the whole car, should be able to quickly and efficiently implement, update and verify the software security components.

5.1.3.2 Production Phase

This phase starts with the actual large-scale production of the car. At this point, no research or development teams work on the conception anymore. The development vehicles have successfully passed all test for functionality, safety and security. During this period, which lasts around 6 years, few minor changes are still possible but may only be taken into account for vehicles that have not been produced yet. Earlier produced vehicles can still be updated through normal maintenance services, e. g., online update or visit to a maintenance garage.

Production Evaluation Criterion: During this phase, car manufacturers should be able to easily reflash the flawed software components and update the distribution security material of the car, e. g., stronger cryptographic keys, new protocols, new security policies. In addition, car manufacturers should still be able to easily update the software and redistribute security material of unpatched cars.

5.1.3.3 Post-Production Phase

This phase ends the actual car production. However the vehicles still on the market have to be maintained and supplied with spare components for both mechanical and electronic purposes. This time period is adjusted depending on the car lifetime, generally around 15 years. Considering the potential of future remote software updates, their lifetime could be significantly expanded. An efficient and durable security management is therefore essential.

Post-Production Evaluation Criterion: During this last phase and like in the previous one, car manufacturers should be able to update the security material. For cost saving, the security mechanisms should even be standardized and reusable for next car generations.

5.2 Middleware

This section provides a description of the two middleware architectures used as implementation bases. The results of their performance evaluations and their first interpretations are then provided. A further discussion about the overall prototype is also presented in Section 5.5.

5.2.1 Etch Middleware

The middleware Apache *Etch* was used as basis for this work implementation. *Etch* is an Open Source Software (OSS) project under the Apache License 2.0, that was extensively investigated as a middleware solution during the SEIS project [177]. According to its online description, “*Etch* is a cross-platform, language- and transport-independent framework for building and consuming network services. [56]” It is common practice in the industry to use/develop an OSS in collaboration with other competitors. First the software is not protected by any restrictive license, which makes its industrial adoption immediate. With a large group of companies using it, the OSS gets more stable, cheaper to maintain and easier to adopt as standard.

Etch Java-binding: Even if the Java-binding of *Etch* was not designed initially for cars, it comes with numerous advantages. It first provides a comprehensive object-oriented IDL and related compiler. They allow to specify in an IDL-based definition file the API used by 2 communicating entities (i. e., client and servers) and generate automatically the middleware skeleton and stub. Then *Etch* is transport-independent and allows flexible RPC mechanisms, i. e., using both TCP and UDP. UDP is actually more efficient on small foot-print platforms, because unlike TCP no channel session must be maintained. A service consumer can directly contact the service provider and directly access the service. *Etch* also features a flexible service management, which leverages both broadcast and central repository for service discovery and sleep-time management [176].

Originally, *Etch* was not developed for a security purpose. The default Java-binding [5] only proposes to establish SSL/TLS channels with an authentication of the server side. With this in mind, the first security extensions have been to provide a mutual authentication with SSL/TLS. The transport features were also extended with a secure “at-least-once” UDP using AES-based encryption and custom authentication schemes based on the EL-Gamal cryptography [22].

Then, the second and main task was to define where to enforce the DIFC or DIFC/DDFT interfaces. For information, the architecture of *Etch* and a short description of its functional and security components are pictured in Figure 5.1. More details about the java binding are not necessary for the general understanding of this thesis. By design, for each new communication session (i. e., one per client socket created, several with a TCP server socket and one with a UDP server socket) a new class of *Filter Chain (FC)* is created. Once initialized, *FC* starts a class *Policy Manager (PM)* (2), which contains all DIFC policies and evaluation mechanisms. For inbound messages, *FC* receives from the *Messagizer* a message payload and its security metadata, e. g., DIFC label, STL taint. *FC* passes these metadata as well as the identity of the requested application to *PM* (2) via the `consult()` function and receives the related policy decision, e. g., *forward* or *not forward* to the application. Inversely for outbound messages, via the `consult()` function, *FC* provides the identity of the sending application and receives the security label or taint to add in the message header. For practical reasons, PMM and SCM are

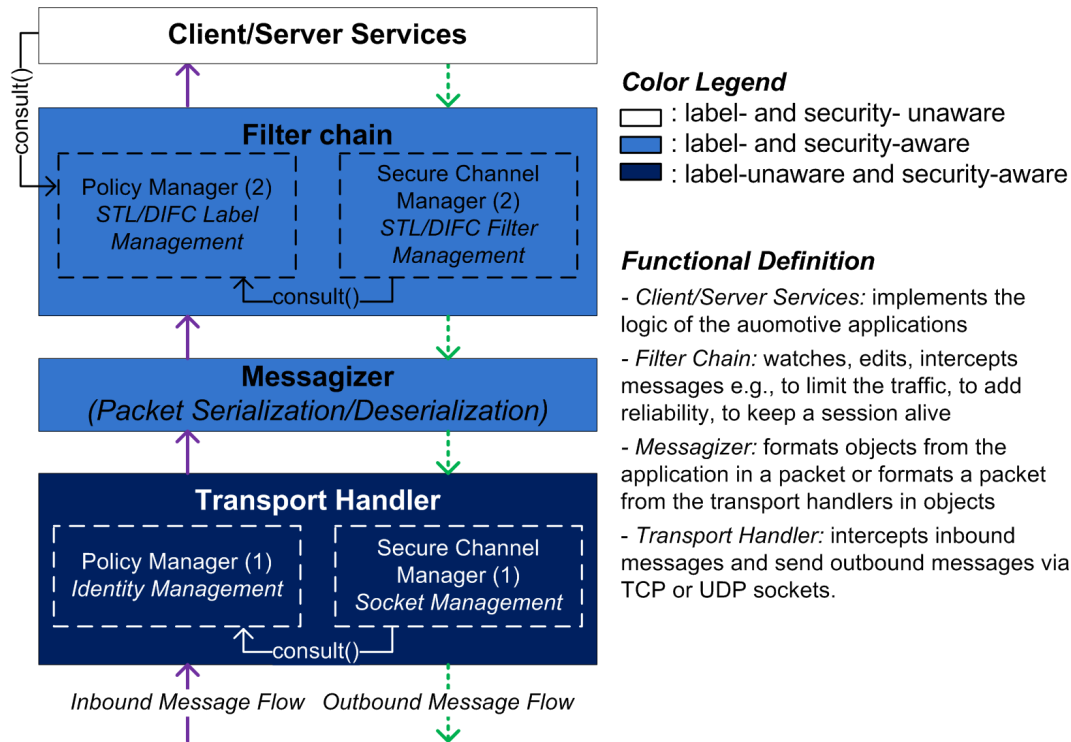


Figure 5.1: Architecture of the *Etch* Java-binding. The figure provides a short description of the functional components and presents a color hierarchy of the components based on their security and IFC awareness. The dashed boxes represent the part of the SME, which have been implemented.

merged. *SCM (2)* is implemented in the *FC* class and enforces the decision of *PM (2)*.

Basically, the *Messagizer* handles security metadata of the header like any other object of the payload. The security part of *Transport Handler*, i. e., the code of *SCM (1)* enforcing the decision of *PM (1)*, only enforces a basic IP address/port filter. ECUs communicate over IPsec together and as a consequence the application is unaware of it. Promising solutions may come with the “Connection Latching” protocol, which makes control information about the IPsec communication available from the application level [185].

Etch C-binding: Because *Etch* originally was not developed for the embedded world, Weckemann et al [179] developed a light C-based and automotive oriented version. C is the programming language of most current automotive applications. This version is optimized for platforms with small performance and is designed to be interoperable with the Java-binding. The main motivation here was to keep this binding simple and not directly map all functionality of *Etch*, but rather just the RPC and notification features over UDP. This version does not provide any dynamic memory management, only static allocations and processor stack pointers have been made available.

For both server and client side, the *Etch* middleware is composed of 3 main components:

- **main.c:** The application logic is coded in this file. As pinpointed in the name, it is the initialization point of the middleware (e.g., launching point of serversocket or socket initialization) and its service. 2 threads are started here, one starting the listening part and the second starting a routine, for example to launch a RPC.
- **etch.c:** This component is actually the logic transforming a C-based object in its binary form and vice versa. The extraction of a first part of the header (Etch function and Function ID) is also performed here.
- **stub.c:** This file includes the “real” middleware part and is where most of the extensions have been performed. Invoked by the **main.c**, the functions of **stub.c** are responsible for the header and payload serialization/deserialization. In addition it is responsible for the socket management, i.e., socket initialization, packet emission and reception.

For this purpose, the **stub.c** was extended with a TCP functionality as well and with the possibility for a server like a HU to establish multiple TCP connections. For this feature, the middleware uses an array of file descriptor describing each new socket. The `select()` function [112] retrieves which entry of the array to use in order to communicate via the right socket. The car architecture is very static. It is therefore easy to start the communication in a specific order and to logically find which socket to communicate with.

Unlike the Java version, the enforcement of DIFC- or STL-based policies is performed directly in the stub during the packet deserialization, as soon as the label field of the header is encountered. For the moment, authorized labels and taints are statically stored in an array and tested for each inbound message with a loop. Future implementation features the use of a hash table instead, which could improve the system efficiency for a large number of labels and taints. Similarly, the labels or taints to add in an outbound message are hard coded in the function responsible for the message serialization.

Etch header serialization: A description of the serialization of the *Etch* packet is presented in Figure 5.2. The *Etch* payload is composed by a series of objects: with the first byte characterizing the data type (e.g., integer, array of characters), the four next ones being a hash value of the object name and the rest being the actual value of the data. Except for the *Etch version* which is directly a usable value, each object requires 5 bytes of “object header”. This characteristic allows a very efficient and robust payload serialization. As a consequence, as soon as the DIFC label or STL taint is extracted, the stub is directly aware of which type of policy to enforce and can if necessary immediately drop the packet before completion. The DIFC label consists of 2 fields, one for the integrity and one for the confidentiality, which can only contain one tag each, whereas the STL taint is about one field of 6 bytes.

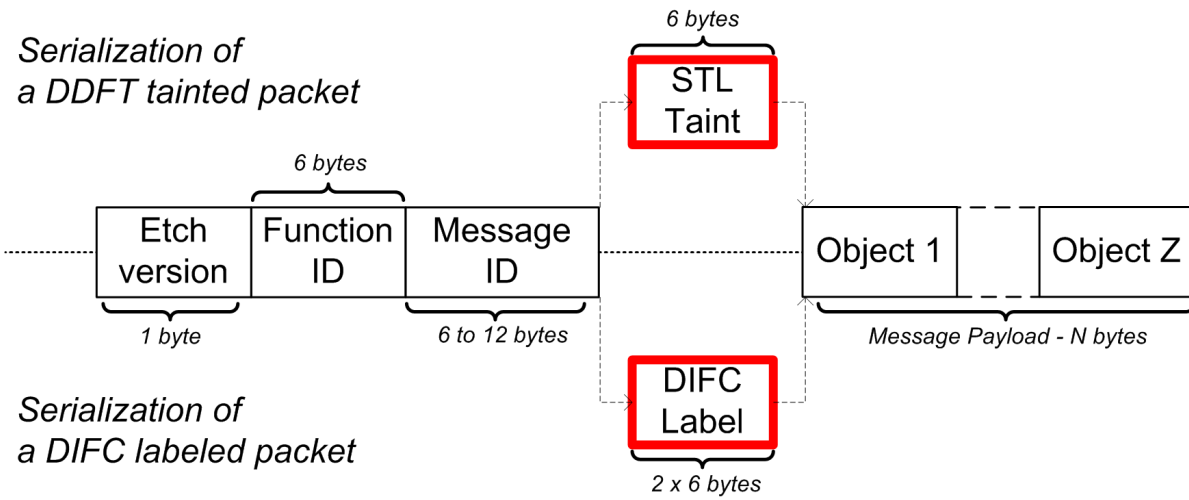


Figure 5.2: Header serialization & in-band protocol of the *Etch* middleware. The lower part of the dashed line represents the *Etch* header with the DIFC-based serialization, while the upper part characterizes the DIFC/DDFT serialization. The parts common to both serializations are in black, the IFC part of the header is in red and specific to the IFC method.

Interface definition: *Etch* provides a compiler, which allows programmers to specify in an IDL-based definition file all function specifications and to have the middleware stub automatically generated. This feature may also be extended with security metadata. These metadata may first configure the secure channel specifications, i. e., whether an authentication is started, whether it is encrypted or whether it requires a specific integrity protection. Some may also take arguments and are an efficient way to specify the DIFC label and ownership of a service, e. g., by directly giving the value of the tags they need to consider. A commented service definition with is given in Listing 5.1.

Regarding the definition of DIFC-based policies, tags are defined for the whole service. The code related to the enforcement of label-based rules is quite repetitive and can automatically be inserted. Only the tag values should be different between the services. Regarding the DIFC/DDFT policies, they are defined in the Chapter 4 and can be specified in a same manner as the tags.

Listing 5.1: Definition of a *Etch* IDL file with specification of security metadata

```
1  /* Definition of the service and its DIFC label */
2
3  @DIFC_integrity(tag_i1)           // definition of the integrity label
4  @DIFC_confidentiality(tag_c1,tag_c2) // definition of the confidentiality label
5  @DIFC_ownership(tag_c3)          // definition of the ownership
6  service Example_Service{
7
8  /* Function of A sending a message to B and returning (receiving from B) and integer */
9  @Direction(Server)               // the message is sent to the server to be processed
10 @Encryption @Authentication      // the communication is encrypted and authenticated
11 int function_1(string message);   // function declaration
12
13 /* Function broadcasting a message */
14 @Broadcast                        // the message is broadcasted
15 @Authentication                  // the message contains an authentication code
16 void function_2(string message);  //function declaration
17
18 /* Function of A sending a message to B and returning (receiving from B) and integer */
19 @Direction(Server)               // the message is sent to the server to be processed
20 @Encryption @Authentication      // the communication is encrypted and authenticated
21 @TPA-enabled(192.168.1.56:5555)  // the function is authorized to communicate with a
22                                 // TPA at the specified IP address
23 @STL_inbound(sl_level,tl_level)  // Inbound STL authorized to be processed
24 @STL_outbound(sl_level,tl_level) // STL to add to an outbound message
25 int function_3(string message);   //function declaration
26 }
```

5.2.2 Performance Results & Interpretation

This first performance test aims at evaluating the average throughput of both *Etch* middleware versions, namely the Java-binding and the C-binding. In this setup, communications are occurring over TCP between two ECUs, i. e., two computers (a server and a client) linked by an Ethernet cable. The client sends a first message containing a buffer of 32 bytes of information and in return expects a buffer of 32 bytes from the server. No computation is performed to generate these buffers. As presented in Section 5.1.1, these measurements have been launched for different types of communication: a) in plaintext, b) over IPsec, c) over IPsec and while enforcing DIFC. The results are presented in Tables 5.1(1) and 5.1(2). Considering that ECUs are mostly using their middleware for control and relatively short data, these tables only present the throughput means and does not propose any bandwidth consideration.

Interpretation: Firstly, both implementations present similar performance. Even if the C-binding has a slightly higher throughput, the Java-binding is a little bit less impacted by the security measures. Unlike the Java *Etch*, the C implementation is a lot simpler, does not perform any dynamic memory management and performs better.

Then the use of IPsec causes the major performance degradation of 26 to 29% and is due to the process of encryption/decryption for each message. The use of specialized HSM benefiting from hardware acceleration capacities may improve the overall performance. The enforcement of DIFC-rules induces an additional and non-negligible degradation of 19 to 21%. This penalty is significant but it can be limited by keeping the

Table 5.1: Throughput performance of the *Etch* middleware: (1) with the Java-binding and (2) with the C-binding. The factor i presents the normalized means with the case a) as reference, the factor ii does the same with the case b). (Enc.: Encryption)

Mode	Throughput			Mode	Throughput		
	(call/sec)	i(%)	ii(%)		(call/sec)	i(%)	ii(%)
a) Blank	1584	100	-	a)	1976	100	-
b) IPsec Enc.	1172	74	100	b)	1402	71	100
c) DIFC+Enc.	950	60	81	c)	1106	56	79

(1) (2)

number of tags low and still provides similar throughput performance as the low-speed CAN. As a first conclusion, the performance penalty is significant. But with the high bandwidth of Ethernet/IP, it is now possible to join small packets that were before carried by several CAN frames (e. g., for an environment model). This results in a reduction in the amount of exchanged traffic and in the need for a very high throughput.

5.3 Security Communication Proxy

This section provides a description of two *Etch* proxy architectures, i. e., java- and C-binding. The results of their performance evaluations and their first interpretations are then provided as well.

5.3.1 Etch Proxy

Like the *Etch* middleware, the *Etch* proxy is equipped with two bindings: one in Java, the other one in C language. The architecture of the Java proxy is depicted in Figure 5.3. The Java version implements all presented features except the STL-based filters. The C-based version is simpler, it does not provide any service discovery feature and communicate with one ECU but multiple external entities. It besides provides an implementation of both DIFC- and STL-based filters

The concept of the proxy is quite straight-forward and based on service mirrors. On-board services propose interfaces to their functions that the proxy can replicate and make available to external devices. The external devices are unaware of the mirrors, which give them the impression to directly communicate with the service. All car functions and services are authenticated only once, when the ECU performs its initial authentication handshake with the proxy. Even if the communications are decoupled, on-board services

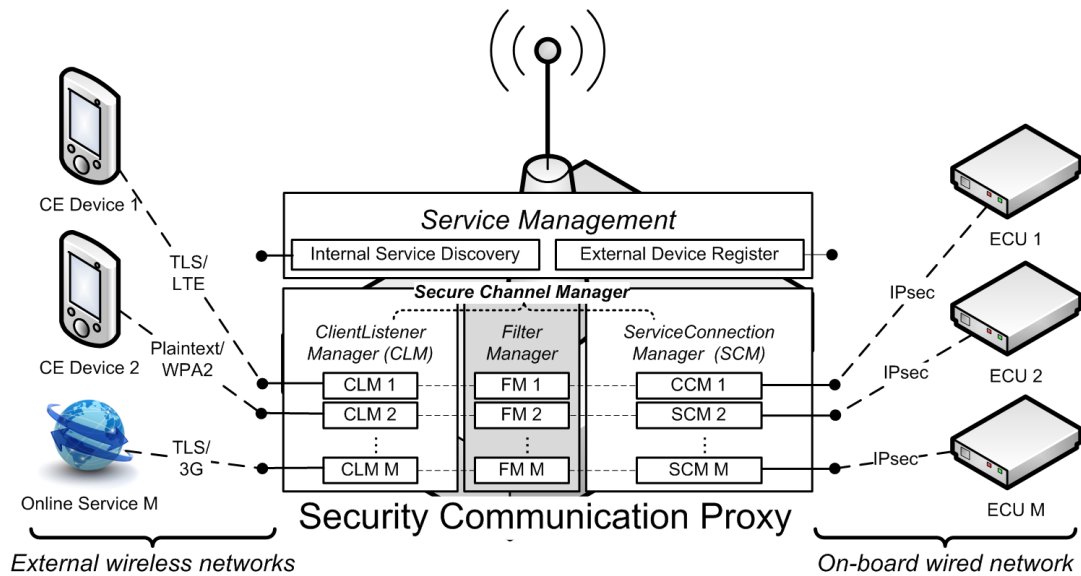


Figure 5.3: Architecture of the *Etch*-enabled communication proxy.

actually get full inbound *Etch* messages extended with a DIFC- and STL-label and can enforce their own security policies.

The proxy is supposed to be installed and merged to the MPA. The proxy has to be aware of all information of both on-board and external communications. Considering that the MPAs still are at the research level. No integration on the physical platform has been possible yet. Instead, the proxy is implemented on a commodity platform like the server and client previously used. All platforms are linked via Ethernet cables.

Etch Java-based Proxy: On the on-board network, both proxy and server are equipped with simple Service Discovery (SD) features. As soon as the server is started, it broadcasts its name and domain. If the proxy did not receive the message, the proxy can also broadcast its presence to discover other servers. For each new SD message specifying a function name and domain, the proxy creates an entry in *ServiceRegistry*.

Regarding the outside, the proxy also provides a SD interface, an external device can contact it and receive a list of all available functions of *ServiceRegistry*. The device chooses the ones it is interested in and announces its choice as well as the protocol (secure or not) it wants to use. In return the proxy answers with an IP address and a port where the device can consume the resource. At this moment the proxy initializes a new *MirrorManager*, which in its turn initializes:

- a *ClientListenerManager*: This class starts the severSocket that the device has to connect to. In addition it provides a queue for inbound messages, so that they all go through the right filter. It also receives messages to directly send to the external device. This class is part of the SCM and may start an SSL/TLS socketServer if required.

- a *ServiceConnectionManager*: This class initializes a socket which get connected to an on-board service. Like the *ClientListenerManager* it provides a queue for outbound messages, so that they all go through the right filter. It also receives messages to directly send to an on-board service. This class is also part of the SCM.
- a *FilterManager*: Between the two previous managers, this class receives a message from one class to the other. After having enforced the adapted IFC filter, this class may pass the message extended or shortened to the *Manager* communicating with the addressee. This class includes the PMM and part of the SCM, i. e., the part enforcing the filters.

Such a dynamic mirror management allows to transparently have multiple services requesting multiple on-board functionalities.

Etch C-based Proxy: This proxy is simpler than the java version. It was built based on a server version of the *Etch* middleware. It is therefore still composed of 3 components. `etch.c` is not modified, neither `main.c`, except that it does not include any application logic (the proxy is application-unaware). Instead, the focus was brought on `stub.c`. Firstly, it initializes now two communication interfaces with two different threads. The first one is a socket accessible by an on-board service, the second one a server socket accessible by an external device. The interfaces are programmed to be cross-oriented, i. e., a packet arriving on one interface is forwarded on the other one. The proxy is not dynamic, therefore it exists twice as many filtering functions on the proxy as there are of available functions on the on-board service. Half of these filtering functions is for inbound messages, the other half is for the outbound ones. Before being forwarded, messages go through their specific proxy filter, DIFC- or STL-based depending on whether a TPA is involved in the communication. Inbound packets are directly extended with the appropriate IFC information and sent to the on-board service. The header of outbound messages is partially deserialized in order to extract the label and enforce a policy. The messages are finally sent to the external device without any IFC information. For the moment only one on-board server can be connected. But it is also possible to extend the proxy, like earlier, with an array of socket descriptors allowing it to receive communications from multiple on-board servers.

5.3.2 Performance Results & Interpretation

This second experiments aims at evaluating the potential throughput of communications occurring through the security communication proxy. Like previously, the communications occur between a server and a client and are traveling through the proxy. The proxy forwards the exchanged messages in both directions. Messages are buffers containing 32 bytes of information and like previously no special computation is done to generate

Table 5.2: Throughput performance of the *Etch* proxy: (1) with the Java-based version and (2) with the C-based version. The factor i presents the normalized means with the case a) taken as reference, the factor ii does the same with the case b). (Enc.: Encryption)

Mode	Throughput		Mode	Throughput		
	(call/sec)	(%)		(call/sec)	i(%)	ii(%)
a) Blank	98	100	a)	1163	100	-
b) TLS/IPsec Enc.	98	100	b)	663	57	100
c) DIFC+Enc.	98	100	c)	632	54	95

(1) (2)

them. Both sides of the communication use TCP. These measurements have been launched for different types of communication: a) blank communications in plaintext, b) with encryption all along the communication, i. e., over IPsec between server and proxy and over TLS between proxy and client, c) with encryption and while enforcing DIFC between proxy and server. The results are presented in Tables 5.2(1) and 5.2(2).

Interpretation: Unlike the previous experiments, the results between the Java- and the C-binding are very different. First regarding the Java-binding, this poor results are due to an implementation issue, that could not be solved. A class of the proxy is responsible to get the *Etch* message from a socket stream and put it in a queue, in order to make sure that all packets go through the DIFC filter. But instead of retrieving a valid *Etch* buffer, the function `read(byte[] b)` of the constructor `InputStream` returns a buffer with only the first byte of the message and causes an error. The implementation is then forced to check the size of the buffer, store it, wait for the rest and re-add the first byte before putting it in the queue before filtering. This bug significantly slows down the communication and causes the bad performance. Otherwise, the addition of security imposes a latency insignificant in comparison to the `InputStream` issue.

Regarding the C-binding, the performance is much better. Like in the first experiments the usage of security protocols is responsible for most of the performance degradation. With such throughput, the addition of DIFC filtering on the proxy and on the server becomes relatively small ($\sim 2\%$). As a conclusion, in comparison to the first experiments, the DIFC enforcement impact seems to be little significant at a lower frequency (~ 600 - 650 calls/sec).

5.4 Monitoring & Controlling the TPA

This section provides a description of two TPA monitoring and controlling methods proposed in this thesis. The results of their performance evaluations and their first

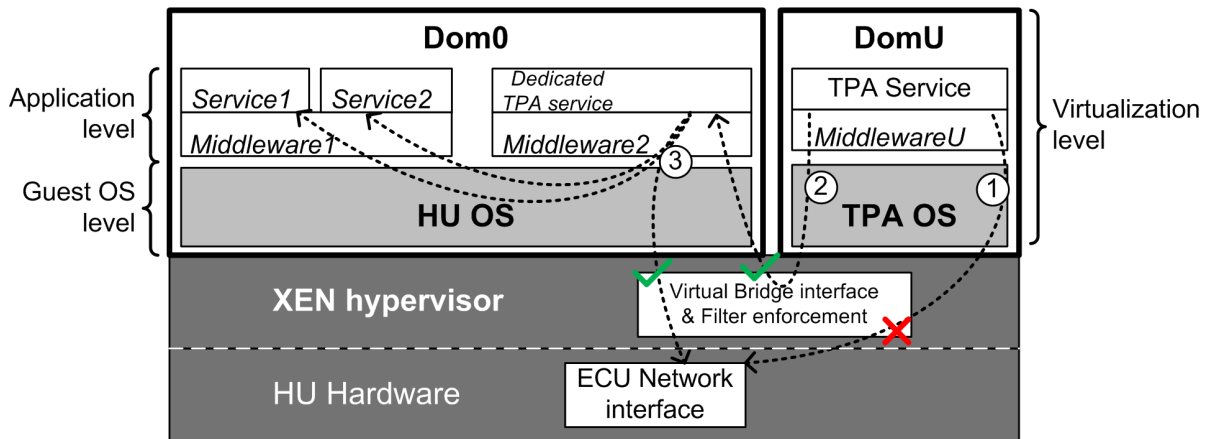


Figure 5.4: Schematic view of a HU architecture leveraging virtualization and middleware for a secure TPA integration. Dashed arrows represent middleware based communications. The communications are numbered as follows:

- (1) from the TPA aiming directly to the on-board network
- (2) from the TPA aiming to the TPA dedicated service of *Middleware2*
- (3) from the dedicated service towards other local services or the network

The red cross characterizes the filters of the hypervisor blocking the communications, whereas the green check shows the same filters letting the communications through.

interpretations are then provided as well.

5.4.1 Isolation and Virtualization

Isolation and virtualization do not allow to directly monitor the TPA. Instead it provides the TPA with a containable execution environment:

- all I/O of this environment, especially the network ones, can be caught and filtered;
- the environment gets allocated a certain amount of ECU resources (e. g., Central Processing Unit (CPU) usage, memory) and cannot access more.

For this implementation, the Xen[®] hypervisor is used [11]. Xen is an open-source type-1 hypervisor, i. e., the hypervisor directly runs on the hardware, controls it and manages the guest OSs running on an upper virtualization level. The integration of the TPA is depicted in Figure 5.4 and shows the architecture of a HU hosting a partition dedicated to the TPA. The trusted HU middleware components developed by car manufacturers, namely *Middleware1* and *Middleware2*, are running in the most privileged domain called Dom0. This domain can freely communicate through the on-board network, i. e., the Xen hypervisor does not enforce any filter. Untrusted TPAs run in an unprivileged cell called DomU on top of a IFC-unaware middleware. Unlike Dom0, DomU is constrained by the

hypervisor and cannot directly contact the on-board network. Instead, Xen implements a virtual bridge filtering all traffic between DomU and Dom0. Only messages between the *Dedicated TPA Service* and DomU are authorized.

The actual IFC enforcement is performed by the *Dedicated TPA Service*. This service enforces the DIFC rules and consists of several functions redirecting messages towards DomU and messages from DomU. For messages to DomU, this service extracts the DIFC label. If the cell is blank, i. e., did not previously receive any message, it labels the cell according to the secrecy tag of the message and forwards it to DomU (i. e., via DLA), otherwise it only forwards a message, if its secrecy label matches the label of the cell (i. e., via the enforcement of the partial order \prec). For messages from DomU, it labels the message according to the cell DIFC label. By default this service does not enforce any rule on integrity labels. It labels messages from DomU with the integrity tag of the cell, i. e., with an untrusted integrity label.

For this implementation, the 4.2 version of Xen is used and the DomU runs a Debian 6.0 OS with 256 MB of allocated RAM. The hypervisor solution was chosen due to its availability and simplicity of usage. However, a similar integration of the dedicated service could have also been performed for a microkernel architecture [91]. A future extension of the hypervisor solution could be to make the middleware DomU-aware. The middleware of the *Dedicated TPA Service* could monitor several statistics of DomU (e. g., memory and CPU usage) thanks to an adapted XenServer [41] API and inform the intrusion detection mechanisms of any TPA misbehavior.

5.4.2 DDFT Engine

The implementation of the DDFT-based IFC approach makes use of a custom version of the *libdft* engine [98]. *libdft* relies on the Intel's Pin [114] for DBI, i. e., in order to inject custom code into an unmodified binary during runtime and enforce predefined policies. In addition, this tool provides an implementation of the shadow memory allowing its efficient management and for predefined machine instructions and systems call to track data flows and propagate some taints accordingly. It also raises warnings or stops the runtime in case of unauthorized behavior and provides better performance in comparison of several other DDFT frameworks [156].

The *libdft* library which is attached as a Pin tool during execution is initially composed of a “main” routine `libdft-dta.c` and of core files like `libdft-core.c` and `tagmap.c`. Originally *libdft* provides a binary tainting, i. e., one byte of memory being tainted by one bit of shadow memory. But the expressiveness of *libdft* got extended in order to fit the STL-based tainting approach, i. e., one byte of memory being tainted by 4 bits, and now expresses all combinations of the STL taxonomy. These modifications concern mostly the core mechanisms of *libdft*. The second set of modifications concerns more the function call monitoring, which is performed in the “main” component of the engine. These extensions/modifications are better described in regard to the source file where they were performed:

- `tagmap.c` specifies the whole tagmap management. The tagmap or shadow memory is the core data structure of *libdft* and is initialized as a bitmap, which keeps the taint information for the virtual address space of a process, e.g., the TPA. The main modifications concern increasing the size of the bitmap, adapting each function tagging a specific portion of bitmap (e.g., taint one, 2, 3, 4, 5, 6, 7, 8 or N bytes) with a specific STL value.
- `libdft-core.c` implements the whole taint propagation logic between registers and memory locations. It uses the tagmap defined earlier and its management functions. The bigger modifications of the tool consist in changing the binary taint propagation into a 4-bit taint propagation. Each taint map of each register has been extended and can be copied to the next map, when the Pin tool detects that the data of a register are copied to another one.
- `libdft-dta.c` provides the implementation of all security policies enforced when a system call (e.g., standard I/O) or a machine instruction is detected. The machine instructions are mainly concerning the *jmp* and *ret* call. Their instrumentation allows to check whether tainted data are jumping in a predefined critical section or whether they are performing a *register assertion*, i.e., writing new data over a register. Pin also provides hooks that allow to catch some system calls based on their IDs. A system-call-specific policy can be apply, e.g., taint or untaint a memory location. This functionality got extended in order to differentiate keyboard inputs from file inputs based on their system call ID. In order to provide more granularity of enforcement, the custom *libdft* directly instruments the functions within the middleware and can stop the TPA execution, if anything goes against one of its policies. As a consequence, when the TPA accesses a file, the argument of `fopen()` are checked to verify if the TPA is allowed to access it in writing and/or in reading. When the TPA reads from a file with `fread`, the received buffer is immediately tainted according to the file sensitivity. When the TPA writes into a file, *libdft* makes sure that the taint of the data matches the taint of the file. When the TPA receives some data through a socket with `read()` for a TCP connection, *libdft* directly extracts the STL taint and directly taints the payload location. Finally when the TPA sends data with `write()`, the TPA middleware serializes the header with some dummy taints. But before being sent, *libdft* intercepts the call and injects an appropriate STL taint.

5.4.3 TPA monitoring evaluation

This section provides the performance results of 3 test scenarios making use of secure TPA monitoring techniques. These scenarios characterize 3 situations relevant to investigate: (1) the *Client-Server* scenario in order to determine the impact of DDFT on a simple bidirectional communication, (2) the *CE device-Proxy-HU-TPA* scenario

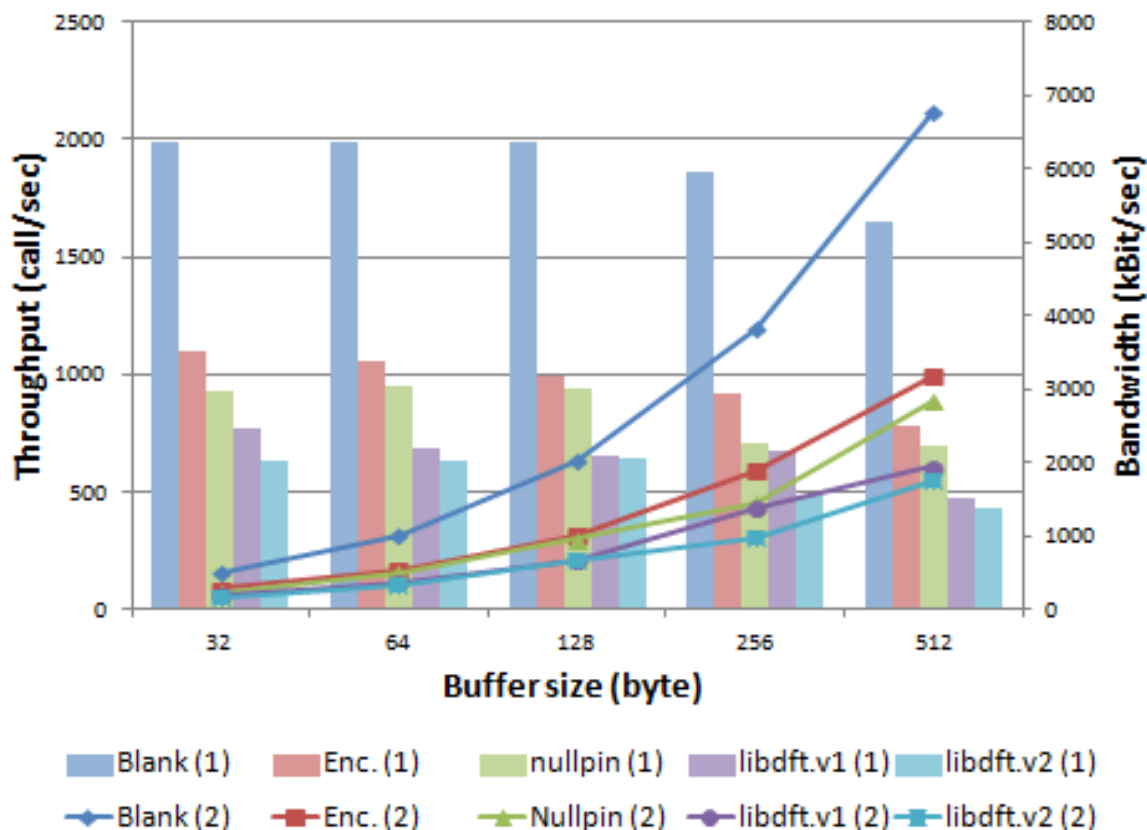


Figure 5.5: Throughput and bandwidth performance of the Client–Server scenario. Measures are provided for several message sizes. The buffer sizes of the messages are specified by the X-axis. Throughput measures are shown by the clustered columns with their Y-axis on the left and commented with (1) in the legend. Bandwidth measures are given by the lines with markers with their Y-axis on the right and commented with (2) in the legend.

to compare the pure DIFC-based and DIFC/DDFT-based approaches, and (3) the *CE device–Proxy–TPA* scenario to evaluate the DDFT monitoring for TPA-based C2X communications. The DDFT performance is dependent on the amount of data processed. As a consequence, the bandwidth and throughput results for various message sizes may be also interesting to investigate. Each scenario is described as follows: a) the experiment, b) a graph of the achieved throughput and bandwidth averages, c) a table presenting the normalized throughput averages independent of any buffer size consideration.

5.4.3.1 Client–Server Scenario

This simple scenario stages a client sending a request to a server which plays the TPA role and waits for an answer. The request consists of a buffer of 30 bytes including

Table 5.3: Normalized throughput performance of the Client–Server scenario. Factor (i), (ii), (iii) and (iv) take respectively (1), (2), (3) and (4) as reference.

	Null (1)	Enc. (2)	Nullpin (3)	libdft.v1 (4)	libdft.v2 (5)
Factor (i)	1	0.51	0.44	0.34	0.29
Factor (ii)	-	1	0.87	0.67	0.58
Factor (iii)	-	-	1	0.77	0.67
Factor (iv)	-	-	-	1	0.87

a header with STL taint and a payload of 2 integers setting the size and type of the answer. The answer is computed by integrating the 2 integers of the request into a table of the requested size. The answer message is therefore about $22 + N$ bytes, with N being the requested length and 22 being the message header size. This experience aims at stressing both middleware and DDFT mechanisms, i. e., taint propagation and system call instrumentation.

The measurements have been performed for different security situations and various sizes of exchanged messages: (1) *blank* considers communications without security, (2) *Enc.* adds the encryption to the communication link, i. e., IPsec, (3) *Nullpin* characterizes the case where the server runs through Pin, but without any DDFT policy enforced, this measurement provides the performance offset of the DBI framework, (4) *libdft.v1* presents the server monitored by the original version of *libdft*, i. e., with binary tainting and custom system call instrumentation, (5) *libdft.v2* is similar to the previous case but the monitoring is done with a *libdft* with STL-based tainting. The last two cases allow to measure the impact of the *libdft* library but also to compare their tainting mechanisms. The results can be found in Figure 5.5 and their normalized averages in Table 5.3.

Interpretation: Table 5.3 shows that the encryption is responsible for the most significant performance decrease. The addition of the Pin tool adds another layer of latency to the system ($\sim 13\%$ of decrease). After each instruction, Pin verifies if there is no policy to enforce and compiles on-the-fly the code to run until the next policy check. The performance decrease induced by the DDFT engine is then due to the instrumentation of the network system call and the taint propagation mechanisms. An increase of the number of taints increases the performance degradation by 13%. Unlike the original *libdft*, the custom *libdft* overwrites the taint value in the shadow memory, instead of just flipping taint bits. Despite the relatively bad performance of the DDFT (34% and 29%), the impact of the DDFT is relatively constant for payload under 128 bytes. The system can still achieve a similar performance to CAN [132] with the Ethernet/IP flexibility.

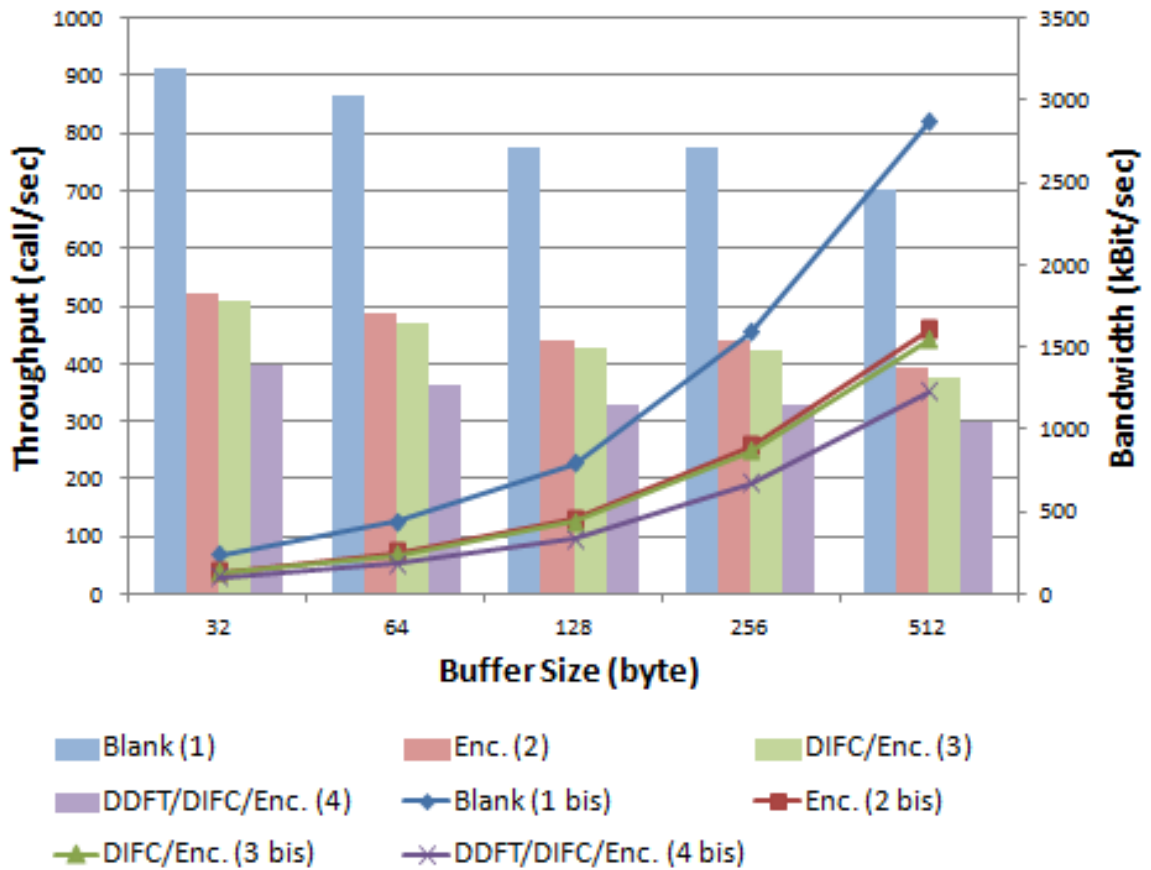


Figure 5.6: Throughput and bandwidth performance of the CE device–Proxy–HU–TPA scenario. Measures are provided for several message sizes. The buffer sizes of the messages are specified by the X-axis. Throughput measures are shown by the clustered columns with their Y-axis on the left and commented with (1) in the legend. Bandwidth measures are given by the lines with markers with their Y-axis on the right and commented with (2) in the legend.

Table 5.4: Normalized throughput performance of the CE device–Proxy–HU–TPA scenario. Factor (i), (ii) and (iii) take respectively (1), (2) and (3) as reference.

	Null (1)	Enc. (2)	Xen/DIFC/Enc. (3)	DDFT/DIFC/Enc. (4)
Factor (i)	1	0.57	0.55	0.43
Factor (ii)	-	1	0.96	0.75
Factor (iii)	-	-	1	0.78

5.4.3.2 CE Device–Proxy–HU–TPA Scenario

This scenario features a CE device sending messages to a TPA and expecting an answer. The message first goes through the proxy, where it gets extended with a DIFC label. On the on-board network, the inbound message then travels until the HU and gets processed either by a dedicated service of the HU or a DIFC/DDFT interface before being forwarded to the TPA. The message structure is similar to the case *Client–Server*. The answer includes like previously an integer table, whose size was defined by the inbound call. The answer takes finally the same inversed path. The TPA runs on the HU computer, either in a Xen cell or monitored by DDFT or running freely on the HU if not specified.

The measure averages of Figure 5.6 have been computed for several security configurations: (1) for blank communications, (2) for encrypted communications, i. e., SSL/TLS for the link CE device–Proxy and IPsec for the link Proxy–HU, (3) for encrypted communications with enforcement of DIFC-policies on the Proxy–HU link and the TPA running in a Xen cell, and (4) for encrypted communications with enforcement of DIFC-policies on the Proxy–HU link and the TPA being monitored by DDFT. These experiments determine the impact of each security feature for a complex use case and also allow to compare the performance of two IFC approaches. The normalized throughput averages of each security configuration can be found in Table 5.4.

Interpretation: For this use case, the normalized impacts of the different security features are actually constant and independent of any buffer size. Like previously, the impact of the encryption is significant. The case (3) shows a very small impact of the DIFC enforcement when compared to the mode (2) (~4% of degradation). After, the impact due to the DDFT (4) is quite substantial: in average 25% of degradation in comparison to the mode (2). However this setup is not really realistic. With DDFT a TPA and a CE device would directly communicate without any intermediary service. This example is about comparing two IFC approaches for the same setup. Based on the results, the Xen/DIFC-based approach provides better performance with up to 500 call/sec and 1500kBit/sec.

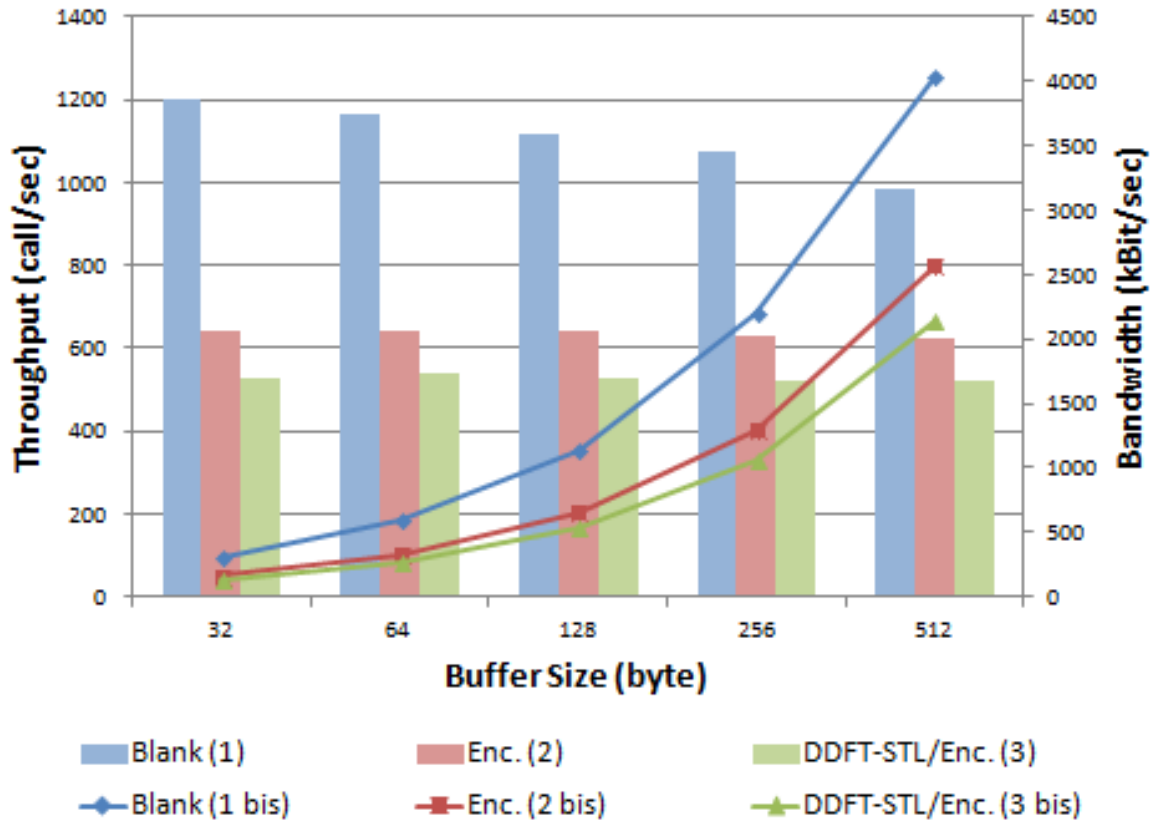


Figure 5.7: Throughput and bandwidth performance of the CE device-Proxy-TPA scenario. Measures are provided for several message sizes. The buffer sizes of the messages are specified by the X-axis. Throughput measures are shown by the clustered columns with their Y-axis on the left and commented with (1) in the legend. Bandwidth measures are given by the lines with markers with their Y-axis on the right and commented with (2) in the legend.

Table 5.5: Normalized throughput performance of the CE device–Proxy–TPA scenario. Factor (i) and (ii) take respectively (1) and (2) as reference.

	Null (1)	Enc. (2)	DDFT/Enc. (3)
Factor (i)	1	0.57	0.47
Factor (ii)	-	1	0.83

5.4.3.3 CE Device–Proxy–TPA Scenario

This last scenario features “direct” communications between CE device and TPA, without any intermediary service of the HU. This scenario can only be performed with DDFT. A CE device sends to the TPA a message similar to the two previous cases and expects the same kind of answer. The messages go through the proxy, where they get extended by a STL taint. The answer messages take the exact same inverted path.

Like the previous scenarios, the measurements were performed for different security configurations: (1) for blank communications, (2) for encrypted communications, i. e., SSL/TLS for the outside IPsec for the inside, and (3) for encrypted communications with enforcement of STL-policies on the Proxy and the TPA being monitored by DDFT. The results can be found in Figure 5.7 and their normalized averages in Table 5.5.

Interpretation: The impacts of the different security features are not surprising and quite similar to what was seen for the last scenario: the encryption and DDFT impacts are significant. But the experiment interest comes actually from the real values of the achieved performance. First, the throughput averages of the measurements (3) stay constant and independent from the buffer size at around 530 calls/sec. In comparison to the measurements (3) of the previous scenario, the throughput values given by the measurements (3) are between 3% (for buffer of 32 bytes) and 38% (for buffer of 512 bytes) higher and therefore achieve a higher bandwidth as well. As conclusion, this last scenario can be considered as the most suitable for direct and TPA-based C2X communications.

5.5 Discussion

This section provides a discussion about implementing security and especially IFC methods at the middleware level and its evaluation. The discussion is segmented in four parts: first (1) about the development of security within the automotive middleware, then (2) about the measured middleware performance, after (3) about IFC recommendations, i. e., which IFC approach to adopt for which use cases, and finally (4) about some limitations of this evaluation.

Security Middleware Development: Developing security at the middleware layer fits the requirements of Section 5.1. The definition of security interfaces directly in the IDL-based file confirms that a small security team can have an overall but relatively abstract view of the car security and still to be able to define all security interfaces, i. e., required mechanism, security enforcement and decision points. In addition, this approach showed to be suitable for systems experiencing demanding testing phases. The automatic generation of a secure stub code and code compartmentalization between secure middleware and application allows to quickly modify the IDL-based file, regenerate the stub without modifying the application logic, re-compile and re-flash the ECU.

Result Analysis & Discussion: The first goal of this chapter was to implement a functional proof-of-concept. While the prototype achieves to implement the concepts presented in Chapters 3 and 4, its evaluation only consists in a first step towards proper validation. For example, this work does not test the suitability of the system for safety-critical mechanisms. In addition to performance benchmarks, further tests should be performed to test the system reactivity and robustness.

Based on these performance results, it can be determined that the usage of security protocols is responsible for the most significant performance penalty. This decrease can however be alleviated by the use of the cryptographic hardware acceleration of adapted HSMs. Nonetheless, the usage of security protocols is necessary. No performance gain should be obtained by letting some communications unprotected and as a consequence easy to attack through eavesdropping and packet injection techniques.

After, DIFC-based mechanisms and TPA monitoring techniques are also responsible for a second and significant performance decrease, which may not be suitable for all in-car use cases. However when evaluating security for new technologies, the focus should not only be on the performance loss induced by the addition of security but also on the measured values. For all evaluated scenarios and with a “full” security package, the secure middleware showed to provide a high throughput and modest bandwidth, i. e., more than 950 calls/sec and 2,3 Mbits/sec for on-board scenarios and 450 calls/sec and 2,1 Mbits/sec for C2X scenarios.

Then, it seems clear that middleware-based techniques will not be used for pure infotainment purposes, which generally require a very big bandwidth, e.g., audio/video or picture transport. For these use cases, Ethernet-based protocols were recommended and allow to save the latency induced by the IP header. As a consequence, the middleware should not be directly compared to the MOST technology, but rather to buses used for control data like CAN. Although these results showed a bandwidth higher than with CAN-based systems, the panel of tested buffer sizes was limited due to some implementation reasons. The C-based middleware used in this work was optimized for relatively small packet and could only process dynamic arrays containing 512 bytes of information. Next versions will also be optimized for bigger buffers and may even perform better.

Additional results not reported here but performed with the same setup and security middleware have already been showing better bandwidth performances with larger static

arrays (1 kByte) but with poor throughput. The scenario client–server with security (Encryption and DIFC) could reach 5,9 Mbits/second and confirms the improvement potential of middleware-based solutions. With the same use of static big arrays, the scenarios CE device–proxy–HU–TPA with DIFC & isolation and CE device–proxy–TPA with DDFT could reach respectively 2,9 Mbits/sec and 4,9 Mbits/sec. The implementation can therefore be considered as suitable for use cases requiring relatively important bandwidth, e. g., car customization, control of cabin features, retrieval of car statistics.

IFC Recommendations Chapter 4 demonstrated that DDFT was providing high flexibility and fine-grained security enforcement and was a more suitable and secure solution than the DIFC/XEN-based one. However some benchmarks tend to show that isolation and DIFC may have better performance for some C2X cases. Besides, whereas the complexity of an application does not influence the performance of an isolated environment, the performance of a big application gets greatly impacted when using DDFT. Tests performed with *libdft* for bigger applications like a web-browser [98] or a MP3-player [156] showed more significant latency (up to 28 times slower). As a consequence, complex TPA or whole untrusted OS requiring the information of a unique user should use DIFC/isolation-based approach. Otherwise, applications requiring multiple inputs from different sensitivity should be monitored with fine-grained DDFT techniques. But for an optimal performance of the DDFT approach, the TPAs should remain light and simple and maximize the use of “trusted”, i. e., non-monitored, libraries.

Additional Limitations: First, the hardware of these experiments was previously and partly used for the evaluation of the original C-based middleware extended by this work [179]. However it cannot account for the performance of a secure middleware installed on a resource-limited platform. It only gives an idea about performance for use cases involving relatively powerful platforms like the HU, the proxy and a CE device, which nonetheless represent a significant part of the use cases for infotainment and control data.

Finally, all measurements were performed in a small 2-to-3-node network involving very simple applications in order to maximize the middleware stressing. However, the car is a system of systems involving a plethora of technologies that were not tested here. A proper and final validation of the secure and IFC-based middleware should include the evaluation of real-car applications in “real-world” situations, e. g., in larger on-board networks generating more “noise” traffic.

About the Use Case of Section 2.3.3: The main question of this use case is about the monitoring method of the “my Driving Coach” TPA. Considering the diversity of inputs, a DDFT seems to be preferable from a pure security point of view. Besides, the TPA is not involved in any time-critical use case, seems relatively simple and light and does not require to run in a specific runtime environment, e. g., like Android. Therefore this TPA should be efficiently monitored by DDFT.

5.6 Summary

This chapter presented the implementation and functional evaluation of a security proof-of-concept including (1) an IP-based DIFC-enabled middleware *Etch* (2) its associated communication proxy and (3) methods for the enforcement of two IFC approaches, namely DIFC/TPA isolation and DIFC/DDFT.

In addition to enhance the security standards of the on-board network, the security middleware provides performances similar to actual automotive bus technology for control data like CAN. When tested for C2X scenarios, the security middleware, proxy and TPA monitoring mechanisms showed to fulfill the needs of use cases requiring a relatively big bandwidth (2,3 Kbits/sec) and a high throughput (~ 450 calls/sec). Recommendations about which IFC solution to use have been proposed as well: DIFC/TPA-isolation for large applications and “untrusted” OS used by a unique user and DIFC/DDFT for lighter applications requiring multiple inputs from different sensitivity.

More than just evaluating the security framework in some particular scenarios, this chapter demonstrated the suitability of implementing security directly at the middleware layer. The IDL-based definition of all security interfaces and the automatic code generation of the policy decision and enforcement points fit particularly the modular approach of the software development and the iterative testing process of the automotive world.

Conclusion and Outlook

This Chapter closes this thesis. The contributions and results are summarized and the conclusions are given in Section 6.1. Finally, an outlook on remaining problems and implications that arose from this work are provided in Section 6.2.

6.1 Summary and Conclusion

For this thesis, several concepts and methods for middleware-based security on future on-board networks were developed. First, the current car architecture as well as its evolution were analyzed in Chapter 2. This phase allowed to identify several necessary key factors for securing forthcoming on-board architectures. The rest of this section proposes to summarize the contributions of this thesis and to highlight how they answer the goals and requirements announced in Chapters 1 and 2.

Security middleware architecture The arrival of Ethernet and IP into cars arises several questions. Of course, the future on-board applications will benefit from the augmented bandwidth and security possibilities. Mature security protocols guaranteeing authentication and encryption will be indeed instantly applicable. But even if it will immediately prevent attackers to eavesdrop any communication or insert a new on-board ECU, it will not be sufficient to solve all automotive security and privacy issues.

The middleware plays a central role in the communication management and is also suitable to abstract the security interfaces. This thesis presents a middleware architecture allowing a consistent on-board security management at several levels of the communication stack and of the resource access. Presented as an extension, this architecture does not depend on a specific middleware implementation.

Regarding the security goal, middleware-based security cannot protect alone against all security attacks of Section 2.3. But thanks to concrete mechanisms leveraging future automotive HSMs and automating the establishment of secure communication channels, the middleware provides a secure way to define a distributed access control within the car and may solve the remaining security issues. At a local level, secure coding practices

limit the risk of security exploits resulting from stack pointer overwriting attacks. But most important, the middleware allows to automate the policy management, setup and enforcement. These enforcement and decision points consist of abstract interfaces allowing to recognize exchanged data as being damaging for the system integrity or as being confidential and can invoke security mechanisms handling them in a proper manner. The rules defining these integrity and confidentiality considerations are defined in the next contributions.

Regarding the functionality goal, the security middleware provides better performance than CAN, but worse than MOST and still allows its application layer to fulfill its mission for safety feature and data control (Requirement 1 in Section 2.4). The middleware modularization enables flexibility of use and provides three security versions that can be adapted to the ECU capacity (Requirement 2). Additionally, its configuration partly static, partly dynamic allows to optimize the settings for car startup-delay and power-saving (Requirements 3 and 4).

Regarding the engineering goal, the automatic generation of secure communication stub and of security enforcement code allows an efficient and consistent software deployment, easy to correct, verify and maintain (Requirement 6). This flexibility allows to manage the security code all along the car life span independently of the application running above (Requirements 7 and 8).

Security communication proxy The customizable and non-regulated nature of CE devices and other online services also raises numerous security concerns about how information may impact the car or the driver. The security proxy, proposed in this thesis, is located on the edge of the network just under the car Multi-Platform Antenna (MPA) and aims at mitigating the C2X risks. The approach basically consists in decoupling all C2X communications. External communication partners are unaware of it, may use a wide range of communication protocols and are not required to follow any security guideline (Requirement 7). In order to qualify these partners and the communication situations, a taxonomy was developed and quantify their Security & Trust Level (STL). Internally, a unique secure middleware-based protocol is used. The proxy enforces domain- and IFC-based filtering on the whole C2X traffic, evaluates on-the-fly the STL of the inbound traffic and therefore supports the car integrity protection. The STL information is carried internally via a middleware-based in-band protocol and allows every ECU to take holistic security decisions as for processing a received message. Inversely, the proxy enforces on outbound traffic filter rules stopping all confidentiality attacks. Finally the centralization of all C2X communications around the proxy and its MPA allows a flexible integration of new standards and STL update (Requirement 5).

Automotive Decentralized Information Flow Control (DIFC) model Securing the communications between two on-board platforms thanks to authentication and encryption is essential, but a security model for function access and data management is still required. This thesis proposes to solve this point with a formal authorization model based on DIFC. Function invocations and data exchanges are abstracted as in-

formation flows. The control over these flows is enforced distributedly at the on-board middleware layer and on the proxy when these flows are entering or leaving the car. The security here relies on methods for labeling the asset to protect, e. g., a function, an application, a database, an exchanged message.

This model focuses both on the integrity and confidentiality of the asset. Instead of enforcing a specific rule for each asset, only two types of policies are necessary, the policy decisions are based on conditions on labels. This model is multipurpose and allows security-consistent and controlled data exchanges between ECUs. As a consequence, the access to critical functions depends on the integrity level of the sender and limits the attack capacity of an attacker. Regarding the confidentiality attacks, confidentiality labels allow services and proxy to be aware of the data sensitivity and to eliminate the risk related to information leak. At a functional level, DIFC has been demonstrated to add an acceptable latency with performance similar to CAN (Requirement 1).

Additionally, DIFC enables a secure integration of Third-Party Applications (TPAs) and external communications partners. Nonetheless, DIFC requires in addition an isolation environment to contain the TPA like XEN and to monitor its I/Os. Like the proxy, this TPA integration allows TPAs to be security-unaware and therefore to simplify their development (Requirement 7)

Automotive Dynamic Data Flow Tracking (DDFT) techniques A deeper and more dynamic TPA integration requires higher security standards in order to protect the car integrity and the information it contains. This thesis proposes to leverage DDFT techniques to secure this use case and its communications. First, when well configured, DDFT eliminates the risk of attacks based on stack pointer overwriting. Then all along the execution, the engine taints TPA inputs according to their STL sensitivity, follows their propagation and can determine the output sensitivity. In addition to control the file access, it enables a real security coupling over the middleware. Via dynamic binary instrumentation, TPAs can only access part of the API and of the on-board resources. This “whitelisting” prevents direct integrity attacks on critical assets. Regarding the confidentiality, the DDFT engine extracts the STL information of any network input and injects a STL into all outputs. Direct communications between TPAs and external environment are filtered based on their STL at the proxy level. For communications occurring between a TPA and an on-board function, an interface on the function side enables a coupling between STL- and DIFC-based enforcements. While having shown some limitations in term of performance, the DDFT techniques are still suitable for light TPAs, which are supposed to be mostly infotainment oriented and not taking part in any time-critical use case. Like the proxy and the DIFC approach, DDFT allows TPAs to be security-unaware and simplifies their development (Requirement 7).

Conclusion In conclusion, this thesis investigated how to secure the on-board network via its middleware and for this purpose, presented a security architecture combining different information flow control techniques that enhance the on-board security management. All contributions provide solutions to protect the car integrity, to elimi-

nate the risk of information leaks and fulfill the security goal set for this thesis. Then at a functional and engineering level, this security architecture fulfills all requirements of Section 2.4 (Requirements 2, 3 and 4 could only quantitatively be evaluated).

Regarding the IFC approaches, several architectures were assessed 1) a pure DIFC version, 2) a STL/DDFT version and finally 3) a DIFC/DDFT version. While the second lacks formal verification and ease of maintenance, the first and third only differs by the way they integrate TPAs. No clear recommendation could be made; the two approaches are complementary. The integration of a full TPA execution environment will benefit from the pure DIFC approach and a XEN-based isolation, whereas a TPA directly installed on a critical part of an ECU will have to be fully controlled by DDFT techniques. While potentially performing worse, the DDFT-monitored TPA will at least benefit from more flexibility and from a more fine-grained security control.

6.2 Outlook and Implications

Real-world implementation This thesis only partly addresses the implementation of the proposed middleware-based security architecture. Even if the authorization models presented in Chapter 4 were fully implemented and integrated in an automotive IP-based middleware, only parts of the Security Middleware Extension (SME) architecture of Chapter 3 were implemented. The modules for intrusion detection, cryptoservices and key management are still missing. Further implementation of the SME and integration with a HSM should be also performed.

Then, the proof-of-concept presented in Chapter 5 mostly makes use of a light version of the automotive middleware *Etch* and was tested for relevant but limited situations in a quite small network. Besides, the requirement for resource-limited platforms was also only considered qualitatively. Further validation would require to test the security architecture with other versions of the middleware in real conditions, e.g., with automotive-specific platforms and in bigger networks producing more noise traffic.

Migration strategy Ethernet/IP-based communications provide reliable bases for automotive innovation and holistic security. Current automotive platforms are complex systems, perform well and represent significant financial investments. Developing a fully Ethernet-based E/E architecture for the next-coming car generation (i.e., for in a few years) does not seem realistic. As a consequence, a gradual transition to Ethernet/IP in cars is planned to start in 2018 and foresees the cohabitation with other traditional bus technologies [157]. While providing a smooth migration, this cohabitation may let part of the on-board architecture unprotected. New and complementary security mechanisms will be required for both non-IP- and IP-enabled systems in order to detect ongoing attacks and avoid critical functionalities to get compromised. At a functional level, several standardization committees for automotive Ethernet and IP-based middleware were already started. The security ones should follow soon.

Drivers & security-awareness This thesis specified for every policy who de-

finer it and for every security decision whether it is dynamically or statically evaluated. Considering the car functional requirements, a significant number of them are statically coded within the middleware and defined by the car manufacturer. However, for more flexibility some cases may require the intervention of a user, e. g., the driver, who may deliberately give to an online service the access to her private information. In the presented framework, these actions require the driver to declassify her information or to include the online service in a privileged list of the proxy. This work considers such cases but did not elaborate on the mechanisms enabling them, e. g., pop-up windows or customization menus. An additional concern regards the display of security-related information, i. e., how to efficiently and safely inform the driver that she is taking an important security decision. Such display should be easily understandable by the driver and not disturb her driving.

Conclusion This brief outlook shows that additional security aspects are still to be investigated. Finally the next and last remark here concerns the potential impact of the security concepts presented in this thesis. Whereas these middleware-based security mechanisms were developed and assessed for the automotive purpose, the middleware layer shows enough flexibility to be adapted depending on relevant use cases and potential threats. Thus it is realistic to think that such concepts can be extended to other domains presenting demanding security and functional requirements, like trains, planes or smart buildings.

Acronyms

2G Second Generation of Mobile Telecommunication Technology

3G Third Generation of Mobile Telecommunication Technology

4G Fourth Generation of Mobile Telecommunication Technology

A/V Audio/Video

ABS Anti-lock Braking System

ACL Access Control List

ADAS Advanced Driver Assistance and Safety

AES Advanced Encryption Standard

AMM Authentication Management Module

API Application Programming Interface

ASIL ISO 26262 - Automotive Safety Integrity Level

ASN.1 Abstract Syntax Notation One

AVB Audio Video Bridging

BCM Brake Control Unit

BYOD Bring Your Own Device

C2X Car-to-X

CAN Controller Area Network

CE Consumer Electronic

CPU Central Processing Unit

CSM Crypto-Service Module

CUID	Car User IDentity
DBI	Dynamic Binary Instrumentation
DES	Data Encryption Standard
DDFT	Dynamic Data Flow Tracking
DIFC	Decentralized Information Flow Control
DLA	Dynamic Label Assignment
DoS	Denial of Service
DTLS	Datagram Transport Layer Security
E/E	Electrical/Electronic
EAP	Extensible Authentication Protocol
ECC	Elliptic Curve Cryptography
ECU	Electronic Control Unit
ESC	Electronic Stability Control
EU	European Union
EVITA	E-safety Vehicle Intrusion proTected Applications
FC	Filter Chain
FSM	Fail Safe Mode
GPS	Global Positioning System
GSM	Global System for Mobile
HDCP	High-bandwidth Digital Content Protection
HMAC	Hash Message Authentication Code
HSFZ	High Speed Fahrzeugzugang
HSM	Hardware Secure Module
HTTP	HyperText Transfer Protocol
HU	Head Unit

I/O Input/Output

IDL Interface Definition Language

IDM Intrusion Detection Module

IDS Intrusion Detection System

IFC Information Flow Control

IKEv2 Internet Key Exchange version 2

IP Internet Protocol

IPC Inter-Process Communication

IPsec Internet Protocol security

ISO International Organization for Standardization

IT Information Technology

JNI Java Native Interface

JVM Java Virtual Machine

KMM Key Management Module

LHW Local Hazard Warning

LIN Local Interconnect network

LTE 4G Long Term Evolution

MAC Message Authentication Code

MPA Multi-Platform Antenna

MOST Media Oriented Systems Transport

NAT Network Address Translation

NFS Network File System

OBD On-Board Diagnostics

OS Operating System

OSI Open Systems Interconnection

- OSS** Open Source Software
- PCM** Powertrain Control Module
- PKI** Public Key Infrastructure
- PM** Policy Manager
- PMM** Policy Management Module
- POJ** Place Of Jurisdiction
- POSIX** Portable Operating System Interface
- QoS** Quality of Service
- RC4** Rivest Cipher 4
- ROP** Return-Oriented Programming
- RPC** Remote Procedure Call
- RSA** Ron Rivest, Adi Shamir and Leonard Adleman encryption
- RSE** Rear Seat Entertainment
- RSU** Road-Side Unit
- SAL** Security Abstraction Layer
- SCM** Secure Channel Module
- SD** Service Discovery
- SEIS** Sicherheit in Eingebetteten IP-basierten Systemen
- SHA-1** Secure Hash Algorithm 1
- SHE** Secure Hardware Extension
- SL** Security Level
- SME** Security Middleware Extension
- SPF** Single Point of Failure
- SQL** Structured Query Language
- SSL** Secure Socket Layer

TLS Transport Layer Security
STL Security & Trust Level
TCP Transport Control Protocol
TL Trust Level
TP Third-Party
TPA Third-Party Application
TPM Trusted Platform Module
TPMS Tire Pressure Monitoring Sensor
UDP User Datagram Protocol
USB Universal Serial Bus
VPN Virtual Private Network
VM Virtual Machine
VLAN Virtual Local Area Network
VSS Vehicle Speed Sensor
Wi-Fi Wireless Fidelity
WEP Wired Equivalent Privacy
WMA Windows Media Audio
WPA Wi-Fi Protected Access
XACML eXtensible Access Control Markup Language
XSS Cross-Site Scripting

Bibliography

- [1] IEEE Standard for Information technology– Local and metropolitan area networks– Specific requirements– Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications, Amendment 6: Wireless Access in Vehicular Environments. *IEEE Std 802.11p-2010* , pages 1–51, 2010.
- [2] B. Aboba, L. Blunk, J. Vollbrecht, J. Carlson, and H. Levkowitz. Extensible Authentication Protocol (EAP). RFC 3748 (Proposed Standard), June 2004. Updated by RFC 5247. [Online]. Available: <http://www.ietf.org/rfc/rfc3748.txt> [Accessed: 22/08/2013].
- [3] J. Al-Jaroodi and A. Al-Dhaheri. Security Issues of Service-Oriented Middleware. *International Journal of Computer Science and Security Networking*, 11(1):153–160, 2011.
- [4] Aleph One. Smashing The Stack For Fun And Profit. *Phrack*, 7(49), 1996.
- [5] Apache Etch. Apache Etch Downloads. Software release. [Online]. Available: <http://etch.apache.org/downloads.html> [Accessed: 22/08/2013].
- [6] Apple. Bonjour Overview. Apple documentation, 2012. [Online]. Available: <https://developer.apple.com/library/mac/documentation/Cocoa/Conceptual/NetServices/Introduction.html> [Accessed: 22/08/2013].
- [7] Aramis Project. Automotive, Railway and Avionics Multicore Systems. Website. [Online]. Available: <http://www.projekt-aramis.de/> [Accessed: 22/08/2013].
- [8] M. Attariyan and J. Flinn. Automating Configuration Troubleshooting with Dynamic Information Flow Analysis. In *Proceedings of the 11th USENIX Symposium on Operating Systems Design and Implementation: OSDI '10*, pages 237–250, 2010.
- [9] AUTOSAR. Specification of UDP Network Management. Technical Report V2.0.0 R4.0 Rev 3, AUTOSAR Standard, 2011.

- [10] S. Axelsson. Intrusion Detection Systems: A Survey and Taxonomy. Technical Report 99-15, Department of Computer Engineering, Chalmers University of Technology, Göteborg, Sweden, 2000.
- [11] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield. Xen and the Art of Virtualization. In *Proceedings of the nineteenth ACM symposium on Operating systems principles: SOSP '03*, pages 164–177, New York, NY, USA, 2003. ACM.
- [12] D. E. Bell and L. J. LaPadula. Secure Computer Systems: Mathematical Foundations. Technical Report MTR-2547, Vol. 1, MITRE Corporation, Bedford, MA, USA, 1973.
- [13] F. Bellard. QEMU, a Fast and Portable Dynamic Translator. In *Proceedings of the 2005 USENIX conference on USENIX annual technical conference: USENIX ATC'05*, pages 41–41, Berkeley, CA, USA, 2005. USENIX Association.
- [14] L. L. Bello. The Case for Ethernet in Automotive Communications. *SIGBED Review*, 8(4):7–15, Dec. 2011.
- [15] I. Bente, G. Dreo, B. Hellmann, S. Heuser, J. Vieweg, J. von Helden, and J. Westhuis. Towards Permission-based Attestation for the Android Platform. In *Proceedings of the 4th international conference on Trust and trustworthy computing: TRUST'11*, pages 108–115, Berlin, Heidelberg, 2011. Springer-Verlag.
- [16] R. Bharadwaj, M. Born, and R. Schreiner. Secure Middleware for Defence Applications. *Journal for Military Communications*, pages 18–1–18–6, 2006.
- [17] Biba. Integrity Considerations for Secure Computer Systems. *MITRE Corporation's, Technical Report ESD-TR 76-372*, 1977.
- [18] N. Bißmeyer, H. Stübing, M. Mattheß, J. Stotz, J. Schütte, M. Gerlach, and F. Friederici. simTD Security Architecture: Deployment of a Security and Privacy Architecture in Field Operational Tests. In *Proceedings of the 7th Embedded Security in Cars Conference: ESCAR '09*, November 2009.
- [19] Black Hat. Black Hat convention. Conference Website. [Online]. Available: <http://www.blackhat.com/> [Accessed: 22/08/2013].
- [20] BMW AG. Navigation System Professional. Website. [Online]. Available: http://www.bmw.com/com/en/insights/technology/technology_guide/articles/navigation_system.html [Accessed: 22/08/2013].
- [21] E. Bonnert. IDF: Atom-Kraft für BMW und Mercedes. Internet article, 2009. [Online]. Available: <http://www.heise.de/ct/meldung/>

- IDF-Atom-Kraft-fuer-BMW-und-Mercedes-789844.html [Accessed: 22/08/2013].
- [22] A. Bouard. Development of a Security Automotive Middleware, Etch. Master's thesis, TELECOM ParisTech/EURECOM, 2010.
- [23] A. Bouard, D. Burgkhardt, and C. Eckert. Middleware-based Security for Hyper-connected Applications in Future In-Car Networks. *EAI Endorsed Transactions on Mobile Communications and Applications*, 13(3), 12 2013.
- [24] A. Bouard, B. Glas, A. Jentsch, A. Kiening, T. Kittel, F. Stadler, and B. Weyl. Driving Middleware Towards a Secure IP-based Future. In *Proceedings of the 10th Embedded Security in Cars Conference: ESCAR '12*, 2012.
- [25] A. Bouard, B. Glas, A. Jentsch, A. Kiening, T. Kittel, F. Stadler, and B. Weyl. SEIS AP4.3: Sicherheit auf Ebene der Applikation und ihrer Middleware. Technical Report TUM-I1215, SEIS Project, 2012. [Online]. Available: <http://mediatum.ub.tum.de/?id=1114356> [Accessed: 22/08/2013].
- [26] A. Bouard, M. Graf, and D. Burgkhardt. Middleware-Based Security and Privacy for In-car Integration of Third-Party Applications. In C. Fernández-Gago, F. Martinelli, S. Pearson, and I. Agudo, editors, *Proceedings of the 7th IFIP WG 11.11 International Conference on Trust Management: IFIP TM '13*, pages 17–32. Springer Berlin Heidelberg, 2013.
- [27] A. Bouard, J. Schanda, D. Herrscher, and C. Eckert. Automotive Proxy-based Security Architecture for CE Device Integration. In C. Borcea, P. Bellavista, C. Giannelli, T. Magedanz, and F. Schreiner, editors, *Proceedings of the 5th International Conference on Mobile Wireless Middleware, Operating Systems, and Applications: MOBILWARE '12*, volume 65, pages 62–76. Springer Berlin Heidelberg, 2013.
- [28] A. Bouard, H. Schweppe, B. Weyl, and C. Eckert. Leveraging In-Car Security by Combining Information Flow Monitoring Techniques. In *Proceedings of the 2nd International Conference on Advances in Vehicular Systems Technologies and Applications: VEHICULAR '13*. ThinkMind, 2013.
- [29] A. Bouard, B. Weyl, and C. Eckert. Practical Information-flow Aware Middleware for In-car Communication. In *Proceedings of the 2013 ACM Workshop on Security, Privacy Dependability for Cyber Vehicles: CyCar '13*, pages 3–8, New York, NY, USA, 2013. ACM.
- [30] D. Brumley, J. Newsome, D. Song, H. Wang, and S. Jha. Towards Automatic Generation of Vulnerability-Based Signatures. In *Proceedings of the 2006 IEEE*

- Symposium on Security and Privacy: SP '06*, pages 2–16, Washington, DC, USA, 2006. IEEE Computer Society.
- [31] O. Bubeck, J. Gramm, M. Ihle, J. Shokrollahi, R. Szerwinski, and M. Emele. A Hardware Security Module for Engine Control Units. In *Proceedings of the 9th Embedded Security in Cars Conference: ESCAR '11*, 2011.
- [32] S. Bunzel. AUTOSAR - the Standardized Software Architecture. *Informatik Spektrum*, 34(1):79–83, 2011.
- [33] CAN-CIA. CAN Specifications. Website. [Online]. Available: <http://www.can-cia.org> [Accessed: 22/08/2013].
- [34] R. N. Charette. This Car Runs on Code. *IEEE Spectrum*, 2009. [Online]. Available: <http://spectrum.ieee.org/green-tech/advanced-cars/this-car-runs-on-code> [Accessed: 22/08/2013].
- [35] S. Checkoway, D. McCoy, B. Kantor, D. Anderson, H. Shacham, S. Savage, K. Koscher, A. Czeskis, F. Roesner, and T. Kohno. Comprehensive Experimental Analyses of Automotive Attack Surfaces. In *Proceedings of the 20th USENIX conference on Security: SEC '11*, pages 6–6, Berkeley, CA, USA, 2011. USENIX Association.
- [36] W. Cheng, Q. Zhao, B. Yu, and S. Hiroshige. TaintTrace: Efficient Flow Tracing with Dynamic Binary Rewriting. In *Proceedings of the 11th IEEE Symposium on Computers and Communications: ISCC '06*, pages 749–754, Washington, DC, USA, 2006. IEEE Computer Society.
- [37] S. Chong and A. C. Myers. Decentralized Robustness. In *Proceedings of the 19th IEEE Computer Security Foundations Workshop: CSFW '06*, pages 242–256, Washington, DC, USA, 2006. IEEE Computer Society.
- [38] S. Chong, K. Vikram, and A. C. Myers. SIF: Enforcing Confidentiality and Integrity in Web Applications. In *Proceedings of 16th USENIX Security Symposium on USENIX Security Symposium: SEC '07*, pages 1:1–1:16, Berkeley, CA, USA, 2007. USENIX Association.
- [39] M. L. Chávez, C. H. Rosete, and F. Rodríguez-Henríquez. Achieving Confidentiality Security Service for CAN. In *Proceedings of the 15th International Conference on Electronics, Communications and Computers CONIELECOMP 2005*, pages 166–170, Washington, DC, USA, 2005. IEEE Computer Society.
- [40] Cisco. NAC appliance - Clean Access Manager Installation and Configuration Guide, Release 4.9. Cisco documentation. [Online]. Available: http://www.cisco.com/en/US/docs/security/nac/appliance/release_notes/492/492rn.html [Accessed: 22/08/2013].

-
- [41] Citrix. XenServer Software Development Kit Guide - Release 5.5.0 Update 2. Citrix documentation. [Online]. Available: http://docs.vmd.citrix.com/XenServer/5.5.0/1.0/en_gb/sdk.html [Accessed: 22/08/2013].
- [42] E. M. Clarke, Jr., O. Grumberg, and D. A. Peled. *Model checking*. MIT Press, Cambridge, MA, USA, 1999.
- [43] J. Clause, W. Li, and A. Orso. Dytan: a Generic Dynamic Taint Analysis Framework. In *Proceedings of the 2007 International Symposium on Software Testing and Analysis: ISSTA '07*, pages 196–206, New York, NY, USA, 2007. ACM.
- [44] L. A. Cuttillo, R. Molva, and T. Strufe. On the Security and Feasibility of Safebook : a Distributed privacy-preserving online social network. In *Proceedings of the 6th International Summer School on Privacy and Identity Management for Life, PrimeLife/IFIP AICT 320*, 2010.
- [45] M. Dalton, H. Kannan, and C. Kozyrakis. Real-World Buffer Overflow Protection for Userspace and Kernelpspace. In *USENIX Security Symposium: USENIX Security '08*, pages 395–410, 2008.
- [46] B. Davis and H. Chen. DBTaint: Cross-Application Information Flow Tracking via Databases. In *Proceedings of the 2010 USENIX conference on Web application development: WebApps'10*, pages 12–12, Berkeley, CA, USA, 2010. USENIX Association.
- [47] K.-O. Detken, H. S. Fhom, R. Sethmann, and G. Diederich. Leveraging Trusted Network Connect for Secure Connection of Mobile Devices to Corporate Networks. In A. Pont, G. Pujolle, and S. V. Raghavan, editors, *Communications: Wireless in Developing Countries and Networks of the Future - 3rd IFIP TC 6 International Conference, WCITD 2010 and IFIP TC 6 International Conference, NF 2010, Held as Part of WCC 2010*, volume 327, pages 158–169. Springer, 2010.
- [48] T. Dierks and E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.2. RFC 5246 (Proposed Standard), Aug. 2008. Updated by RFCs 5746, 5878, 6176. [Online]. Available: <http://www.ietf.org/rfc/rfc5246.txt> [Accessed: 22/08/2013].
- [49] K. Dietrich. Automotive Security in the Internet of Things and Services (IoTS). Presentation slides (ESCAR '12), 2012. [Online]. Available: https://www.escar.info/fileadmin/Datastore/2012_escar-Vortraege/Dieterich-Bosch-Keynote-Presentation.pdf [Accessed: 22/08/2013].
- [50] D. Dolev and A. C. Yao. On the Security of Public Key Protocols. Technical report, Stanford University, Stanford, CA, USA, 1981.

- [51] K. Edwards, V. Bellotti, A. K. Dey, and M. Newman. Stuck in the Middle: The Challenges of User-Centered Design and Evaluation for Middleware. In *Proceedings of the 2003 Conference on Human Factors in Computing Systems: CHI '03*, 2003.
- [52] P. Efstathopoulos, M. Krohn, S. VanDeBogart, C. Frey, D. Ziegler, E. Kohler, D. Mazières, F. Kaashoek, and R. Morris. Labels and Event Processes in the Asbestos Operating System. In *Proceedings of the 20th ACM symposium on Operating systems principles: SOSP '05*, pages 17–30, New York, NY, USA, 2005. ACM.
- [53] M.-A. Elliott. The Future of Connected Cars. Internet article, 2011. [Online]. Available: <http://mashable.com/2011/02/26/connected-car/> [Accessed: 22/08/2013].
- [54] W. Enck, P. Gilbert, B.-G. Chun, L. P. Cox, J. Jung, P. McDaniel, and A. N. Sheth. TaintDroid: an Information-Flow Tracking System for Realtime Privacy Monitoring on Smartphones. In *Proceedings of the 9th USENIX conference on Operating systems design and implementation: OSDI'10*, pages 1–6, Berkeley, CA, USA, 2010. USENIX Association.
- [55] escript. Efficient Public Key Infrastructure (PKI) Solutions for Embedded Systems, 2012. [Online]. Available: <https://www.escript.com/company/single-news/detail/efficient-public-key-infrastructure-pki-solutions-for-embedded-systems/> [Accessed: 22/08/2013].
- [56] A. Etch. Etch Homepage. Website, 2013. [Online]. Available: <http://etch.apache.org/> [Accessed: 22/08/2013].
- [57] EVITA Project. E-safety Vehicle Intrusion proTected Applications. Website. [Online]. Available: <http://evita-project.org/> [Accessed: 22/08/2013].
- [58] Facebook. Facebook Homepage. Website. [Online]. Available: <http://www.facebook.com> [Accessed: 22/08/2013].
- [59] FlexRay-Consortium. FlexRay specification Electrical Physical Layer. Website. [Online]. Available: http://tge.cmaisonneuve.qc.ca/barbaud/R%C3%A9f%C3%A9rences%20techniques/Protocoles%20C%A0%20tranches%20de%20temps/FlexRay_Electrical_Physical_Layer_Specification_V2.1_Rev.B.pdf [Accessed: 22/08/2013].
- [60] Ford. Ford SYNC Support. Website. [Online]. Available: <http://www.ford.com/technology/sync/> [Accessed: 22/08/2013].

-
- [61] Fujitsu Semiconductor Europe. Fujitsu Announces Powerful MCU with Secure Hardware Extension (SHE) for Automotive Instrument Clusters. Fujitsu press release, 2012. [Online]. Available: http://www.fujitsu.com/emea/news/pr/fseu-en_20121129-1044-fujitsu-mcu-secure-hardware-extension-atlas-1.html [Accessed: 22/08/2013].
- [62] T. Garfinkel and M. Rosenblum. A Virtual Machine Introspection Based Architecture for Intrusion Detection. In *Proceedings of the 10th Annual Network and Distributed System Security Symposium : NDSS '03*, 2003.
- [63] M. Glass, D. Herrscher, H. Meier, M. Piastowski, and P. Shoo. SEIS - Security in Embedded IP-based Systems. *ATZelektronik Worldwide*, 2010-01, pages 36–40, 2010.
- [64] M. G. Graff and K. R. van Wyk. *Secure Coding - Principles and Practices: Designing and Implementing Secure Applications*. O'Reilly Media, Newton, MA, USA, 2003.
- [65] A. Groll and C. Ruland. Secure and Authentic Communication on Existing In-Vehicle Networks. In *In Proceedings of the 2009 IEEE Intelligent Vehicles Symposium: IV '09*, pages 1093–1097, 2009.
- [66] A. Gutowska. Research in Online Trust: Trust Taxonomy as A Multi-Dimensional Model. Technical report, School of Computing and Information Technology, University of Wolverhampton, 2007.
- [67] V. Haldar, D. Chandra, and M. Franz. Practical, Dynamic Information-flow for Virtual Machines. Technical report, Department of Information and Computer Scien, University of California, 2005.
- [68] M. Henning and M. Spruiell. Choosing Middleware: Why Performance and Scalability do (and do not) Matter. White paper, 2011. [Online]. Available: <http://www.zeroc.com/articles/IcePerformanceWhitePaper.pdf> [Accessed: 22/08/2013].
- [69] K. Hess. The Top 5 Trends in Mobile and BYOD Security. Internet article, 2013. [Online]. Available: <http://www.zdnet.com/the-top-five-trends-in-mobile-and-byod-security-7000014226/> [Accessed: 22/08/2013].
- [70] B. Hicks, S. Rueda, T. Jaeger, and P. McDaniel. From Trusted to Secure: Building and Executing Applications that Enforce System Security. In *Proceedings of the 2007 USENIX conference on USENIX annual technical conference: USENIX ATC'07*, pages 16:1–16:14, Berkeley, CA, USA, 2007. USENIX Association.

- [71] M. Hicks, S. Tse, B. Hicks, and S. Zdancewic. Dynamic Updating of Information-Flow Policies. In *Proceedings of the Foundations of Computer Security Workshop*, 2005.
- [72] T. Hoppe, S. Kiltz, and J. Dittman. Sniffing/Replay Attacks on CAN Buses: A Simulated Attack on the Electric Window Lift Classified Using an Adapted CERT Taxonomy. In *In Proceedings of the 2nd Workshop on Embedded Systems Security: WESS '07*, 2007.
- [73] T. Hoppe, S. Kiltz, and J. Dittmann. Adaptive Dynamic Reaction to Automotive IT Security Incidents Using Multimedia Car Environment. In M. Rak, A. Abraham, and V. Casola, editors, *Proceedings of the 4th International Conference on Information Assurance and Security: IAS '08*, pages 295–298. IEEE Computer Society, 2008.
- [74] T. Hoppe, S. Kiltz, and J. Dittmann. Adaptive Dynamic Reaction to Automotive IT Security Incidents Using Multimedia Car Environment. In *Proceedings of the 4th International Conference on Information Assurance and Security: IAS '08*, pages 295–298, 2008.
- [75] T. Hoppe, S. Kiltz, and J. Dittmann. Security Threats to Automotive CAN Networks - Practical Examples and Selected Short-Term Countermeasures. In *Proceedings of the 27th international conference on Computer Safety, Reliability, and Security: SAFECOMP '08*, pages 235–248, Berlin, Heidelberg, 2008. Springer-Verlag.
- [76] T. Hoppe, S. Kiltz, and J. Dittmann. Applying Intrusion Detection to Automotive it - Early Insights and Remaining Challenges. *Journal of Information Assurance and Security*, 4(6):226–235, 2009.
- [77] T. Hoppe, S. Kiltz, and J. Dittmann. Automotive IT-Security as a Challenge: Basic Attacks from the Black Box Perspective on the Example of Privacy Threats. In B. Buth, G. Rabe, and T. Seyfarth, editors, *Proceedings of the 27th international conference on Computer Safety, Reliability, and Security: SAFECOMP '09*, volume 5775, pages 145–158, Berlin, Heidelberg, 2009. Springer-Verlag.
- [78] J. D. Howard and T. A. Longstaff. A Common Language for Computer Security Incidents. Technical Report SAND98-8667, Sandia National Laboratories, Albuquerque, CA, USA, 1998.
- [79] M. Howard and D. Leblanc. *Writing Secure Code*. Microsoft Press, Redmond, WA, USA, 2001.
- [80] M. Huebler. Bmw Connected Drive - Case Study: The Connected Driving Experience. Presentation slides, 2012. [Online]. Available:

- <http://www.apps-world.net/europe/images/stories/presentation2012/dev-d2-1550-michael-huebler-bmw.pdf> [Accessed: 22/08/2013].
- [81] M. S. Idrees and Y. Roudier. Effective and Efficient Security Policy Engines for Automotive On-board Networks. In *Proceedings of the 4th International Workshop on Communications Technologies for Vehicles: Nets4 Cars/Nets4 Trains '12*, 2012.
- [82] IEEE. IEEE 1003.1-2001 Standard for IEEE Information Technology - Portable Operating System Interface (POSIX(R)). IEEE proposed standard, 2001. [Online]. Available: <http://standards.ieee.org/findstds/standard/1003.1-2001.html> [Accessed: 22/08/2013].
- [83] IEEE 802.1 Working Group. 802.1AE - Media Access Control (MAC) Security. Website, 2006. [Online]. Available: <http://www.ieee802.org/1/pages/802.1ae.html> [Accessed: 22/08/2013].
- [84] IEEE 802.1 Working Group. 802.1AS - Timing and Synchronization. Website, 2010. [Online]. Available: <http://www.ieee802.org/1/pages/802.1as.html> [Accessed: 22/08/2013].
- [85] IEEE 802.1 Working Group. Audio/Video Bridging Task Group. Website, 2012. [Online]. Available: <http://www.ieee802.org/1/pages/avbridges.html> [Accessed: 22/08/2013].
- [86] IEEE Standards Association. IEEE 1722 - Layer 2 Transport Protocol Working Group for Time-sensitive Streams. Website, 2011. [Online]. Available: <http://grouper.ieee.org/groups/1722/> [Accessed: 22/08/2013].
- [87] IEEE Standards Association - WG802.1 - Higher Layer LAN Protocols Working Group. 802.1X-2010 - IEEE Standard for Local and metropolitan area networks—Port-Based Network Access Control. IEEE proposed standard. [Online]. Available: <http://standards.ieee.org/findstds/standard/802.1X-2010.html> [Accessed: 22/08/2013].
- [88] A. Ingram. BMW i3 Smartphone App Previews The Future. Internet article, 2012. [Online]. Available: http://www.motorauthority.com/news/1071265_bmw-i3-smartphone-app-previews-the-future [Accessed: 22/08/2013].
- [89] Inova Semiconductors GmbH. APIX2 - Automotive Pixel Link. Website. [Online]. Available: http://www.inova-semiconductors.de/en/products_apix2.html [Accessed: 22/08/2013].
- [90] A. Iqbal, N. Sadeque, and R. I. Mutia. An Overview of Microkernel, Hypervisor and Microvisor Virtualization Approaches for Embedded Systems. Technical report, Department of IET, Lund University, 2010.

- [91] A. Iqbal, N. Sadeque, and R. I. Mutia. An Overview of Microkernel, Hypervisor and Microvisor Virtualization Approaches for Embedded Systems. White paper, 2010. [Online]. Available: www.eit.lth.se/fileadmin/eit/project/142/virtApproaches.pdf [Accessed: 22/08/2013].
- [92] ISO. *ISO/IEC 27002:2005 – Information Technology – Security Techniques – Code of Practice for Information Security Management*. International Organization for Standardization, Geneva, Switzerland, 2005.
- [93] ISO. *ISO/IEC 15408:2009 – Information Technology – Security Techniques – Evaluation Criteria for IT Security*. International Organization for Standardization, Geneva, Switzerland, 2009.
- [94] ISO. *ISO 26262:2011 – Road vehicles – Functional safety*. International Organization for Standardization, Geneva, Switzerland, 2011.
- [95] ISO. *ISO/IEC 15443:2012 – Information Technology – Security Techniques – A Framework for Security Assurance*. International Organization for Standardization, Geneva, Switzerland, 2012.
- [96] V. Issarny, M. Caporuscio, and N. Georgantas. A Perspective on the Future of Middleware-based Software Engineering. In *Proceedings of the Workshop on the Future of Software Engineering : FOSE '07*, pages 244–258, 2007.
- [97] C. Kaufman, P. Hoffman, Y. Nir, and P. Eronen. Internet Key Exchange Protocol Version 2 (IKEv2). RFC 5996 (Proposed Standard), Sept. 2010. Updated by RFC 5998. [Online]. Available: <http://www.ietf.org/rfc/rfc5996.txt> [Accessed: 22/08/2013].
- [98] V. P. Kemerlis, G. Portokalidis, K. Jee, and A. D. Keromytis. libdft: Practical Dynamic Data Flow Tracking for Commodity Systems. In *Proceedings of the 8th ACM SIGPLAN/SIGOPS conference on Virtual Execution Environments: VEE '12*, pages 121–132, New York, NY, USA, 2012. ACM.
- [99] S. Kent and K. Seo. Security Architecture for the Internet Protocol. RFC 4301 (Proposed Standard), Dec. 2005. Updated by RFC 6040. [Online]. Available: <http://www.ietf.org/rfc/rfc4301.txt> [Accessed: 22/08/2013].
- [100] T. Kivinen. Minimal IKEv2 draft-ietf-lwig-ikev2-minimal-00.txt. Technical report, Internet Engineering Task Force (IETF). [Online]. Available: <http://tools.ietf.org/html/draft-ietf-lwig-ikev2-minimal-00> [Accessed: 22/08/2013].
- [101] N. Koblitz. Elliptic Curve Cryptosystems. *Journal of Mathematics of Computation*, 48(177):203–209, 1987.

-
- [102] K. Koscher, A. Czeskis, F. Roesner, S. Patel, T. Kohno, S. Checkoway, D. McCoy, B. Kantor, D. Anderson, H. Shacham, and S. Savage. Experimental Security Analysis of a Modern Automobile. In *Proceedings of the 2010 IEEE Symposium on Security and Privacy: SP '10*, pages 447–462, Washington, DC, USA, 2010. IEEE Computer Society.
- [103] C. Krauß, F. Stumpf, and C. Eckert. Detecting Node Compromise in Hybrid Wireless Sensor Networks Using Attestation Techniques. In *Proceedings of the 4th European Workshop on Security and Privacy in Ad hoc and Sensor Networks: ESAS '07*, volume 4572 of *Lecture Notes in Computer Science*, pages 203–217. Springer-Verlag, 2007.
- [104] M. Krohn, A. Yip, M. Brodsky, N. Cliffer, M. F. Kaashoek, E. Kohler, and R. Morris. Information Flow Control for Standard OS Abstractions. *SIGOPS Operating System Review*, 41(6):321–334, 2007.
- [105] B. W. Lampson. A Note on the Confinement Problem. *Communications of the ACM*, 16(10):613–615, Oct. 1973.
- [106] A. Lang, J. Dittmann, S. Kiltz, and T. Hoppe. Future Perspectives: The Car and Its IP-Address - A Potential Safety and Security Risk Assessment. In F. Saglietti and N. Oster, editors, *Proceedings of the 26th international conference on Computer Safety, Reliability, and Security: SAFECOMP '07*, volume 4680, pages 40–53. Springer, 2007.
- [107] M. Lange, S. Liebergeld, A. Lackorzynski, A. Warg, and M. Peter. L4Android: a Generic Operating System Framework for Secure Smartphones. In *Proceedings of the 1st ACM workshop on Security and Privacy in Smartphones and Mobile Devices: SPSM '11*, pages 39–50, New York, NY, USA, 2011. ACM.
- [108] U. Larson, D. K. Nilsson, and E. Jonsson. An Approach to Specification-based Attack Detection for In-Vehicle Networks. In *In Proceedings of the IEEE Intelligent Vehicles Symposium: IV '08*, 2008.
- [109] H.-T. Lim, L. Völker, and D. Herrscher. Challenges in a Future IP/ethernet-based In-car Network for Real-time Applications. In *Proceedings of the 2011 Design Automation Conference: DAC '11*, pages 7–12, 2011.
- [110] LIN-Consortium. LIN Specification Package Rev. 2.1. LIN documentation, 2006. [Online]. Available: <http://www.lin-subbus.org> [Accessed: 22/08/2013].
- [111] T. C. Ling and et al. *Baker & McKenzie - Global Privacy Handbook*. International Association for Contract and Commercial Management (IACCM), Ridgefield, CT, USA, 2012.

- [112] Linux Programmer's Manual. SELSELECT(2) Man page. Online documentation, 2012. [Online]. Available: <http://man7.org/linux/man-pages/man2/select.2.html> [Accessed: 22/08/2013].
- [113] D. C. P. LLC. High-Bandwidth Digital Content Protection System. Technical Report Revision 1.4, Digital Content Protection LLC, 2009.
- [114] C.-K. Luk, R. Cohn, R. Muth, H. Patil, A. Klauser, G. Lowney, S. Wallace, V. J. Reddi, and K. Hazelwood. Pin: Building Customized Program Analysis Tools with Dynamic Instrumentation. In *Proceedings of the 2005 ACM SIGPLAN conference on Programming language design and implementation: PLDI '05*, pages 190–200, New York, NY, USA, 2005. ACM.
- [115] Z. Lutz. Renault ebuts R-Link, an in-dash Android system with app market. Internet Article, 2011. [Online]. Available: <http://www.engadget.com/2011/12/09/renault-debuts-r-link-an-in-dash-android-system-with-app-market/> [Accessed: 22/08/2013].
- [116] R. C. Mayer, J. H. Davis, and F. D. Schoorman. An Integrative Model of Organizational Trust. *The Academy of Management Review*, 20(3):709–734, 1995.
- [117] B. McCarty. *SELinux: NSA's Open Source Security Enhanced Linux*. O'Reilly Media, Newton, MA, USA, 2004.
- [118] G. McGraw. Software Security: Building Security In. In *Proceedings of 17th International Symposium on Software Reliability Engineering: ISSRE '06.*, page 6, 2006.
- [119] R. McMillan. 'War Texting' Lets Hackers Unlock Car Doors via SMS. Internet article, 2011. [Online]. Available: http://www.pcworld.com/article/236678/War_Texting_Lets_Hackers_Unlock_Car_Doors_via_SMS.html [Accessed: 22/08/2013].
- [120] C. Mecklenbrauker, A. Molisch, J. Karedal, F. Tufvesson, A. Paier, L. Bernado, T. Zemen, O. Klemp, and N. Czink. Vehicular Channel Characterization and Its Implications for Wireless System Design and Performance. *Proceedings of the IEEE*, 99(7):1189–1212, 2011.
- [121] C. F. Mecklenbräuker, A. F. Molisch, J. Karedal, F. Tufvesson, A. Paier, L. Bernadó, T. Zemen, O. Klemp, and N. Czink. Vehicular Channel Characterization and Its Implications for Wireless System Design and Performance. *Proceedings of the IEEE*, 99(7):1189–1212, 2011.

-
- [122] M. Migliavacca, I. Papagiannis, D. M. Eyers, B. Shand, J. Bacon, and P. Pietzuch. DEFCON: High-Performance Event Processing with Information Security. In *Proceedings of the 2010 USENIX conference on USENIX annual technical conference: USENIX ATC'10*, pages 1–1, Berkeley, CA, USA, 2010. USENIX Association.
- [123] V. S. Miller. Use of Elliptic Curves in Cryptography. In *Lecture notes in computer sciences; 218 on Advances in cryptology: CRYPTO '85*, pages 417–426, New York, NY, USA, 1986. Springer-Verlag New York, Inc.
- [124] A. Monot, N. Navet, F. Simonot, and B. Bavoux. Multicore Scheduling in Automotive ECUs. In *Proceedings of the Conference Embedded Real Time Software and Systems: ERTSS '10*, 2010.
- [125] MOST-Cooperation. MOST Homepage. Webstire. [Online]. Available: <http://www.mostnet.de> [Accessed: 22/08/2013].
- [126] T. Murphy. BYOD and Top 5 Network Security Threat for 2012. Presentation slides, 2012. [Online]. Available: <http://www.slideshare.net/BradfordNetworks/byod-and-top-5-network-security-threats-for-2012> [Accessed: 22/08/2013].
- [127] D. Muthukumaran, A. Sawani, J. Schiffman, B. M. Jung, and T. Jaeger. Measuring Integrity on Mobile Phone Systems. In *Proceedings of the 13th ACM symposium on Access control models and technologies: SACMAT '08*, pages 155–164, New York, NY, USA, 2008. ACM.
- [128] A. C. Myers. JFlow: Practical Mostly-static Information Flow Control. In *Proceedings of the 26th ACM SIGPLAN-SIGACT symposium on Principles of programming languages: POPL '99*, pages 228–241, New York, NY, USA, 1999. ACM.
- [129] A. C. Myers, S. Chong, N. Nystrom, L. Zheng, and S. Zdancewic. Jif: Java Information Flow. Software release available at <http://www.cs.cornell.edu/jif/>.
- [130] A. C. Myers and B. Liskov. A Decentralized Model for Information Flow Control. *The SIGOPS Operating System Review*, 31(5):129–142, 1997.
- [131] S. Mysore, B. Mazloom, B. Agrawal, and T. Sherwood. Understanding and Visualizing Full Systems with Data Flow Tomography. In *Proceedings of the 13th International Conference on Architectural Support for Programming Languages and Operating Systems: ASPLOS '08*, pages 211–221, New York, NY, USA, 2008. ACM.
- [132] National Instruments. CAN Physical Layer Standards: High-Speed vs. Low-Speed/Fault-Tolerant CAN. National Instrument documentation, 2002. [Online]. Available: <http://digital.ni.com/public.nsf/allkb/84210794086E9C0886256C1C006BE6AE> [Accessed: 22/08/2013].

- [133] M. Nauman, S. Khan, X. Zhang, and J.-P. Seifert. Beyond Kernel-level Integrity Measurement: Enabling Remote Attestation for the Android Platform. In *Proceedings of the 3rd international conference on Trust and trustworthy computing: TRUST'10*, pages 1–15, Berlin, Heidelberg, 2010. Springer-Verlag.
- [134] N. Navet, A. Monot, B. Bavoux, and F. Simonot-Lion. Multi-source and Multicore Automotive ECUs - OS Protection Mechanisms and Scheduling. In *Proceedings of the 2010 IEEE International Symposium on Industrial Electronics: ISIE '10*, pages 3734–3741, 2010.
- [135] N. Nethercote and J. Seward. Valgrind: a Framework for Heavyweight Dynamic Binary Instrumentation. In *Proceedings of the 2007 ACM SIGPLAN conference on Programming language design and implementation: PLDI '07*, pages 89–100, New York, NY, USA, 2007. ACM.
- [136] J. Newsome and D. Song. Dynamic Taint Analysis for Automatic Detection, Analysis, and Signature Generation of Exploits on Commodity Software. In *Proceedings of the 12th Annual Network and Distributed System Security Symposium : NDSS '05*, 2005.
- [137] D. K. Nilsson, U. Larson, and E. Jonsson. Efficient In-Vehicle Delayed Data Authentication Based on Compound Message Authentication Codes. In *Proceedings of the 68th IEEE Conference on Vehicular Technology: VTC '08-Fall*, pages 1–5, 2008.
- [138] Object Management Group. Real-time CORBA Specification. OMG documentation, 2005. [Online]. Available: <http://www.ois.com/images/stories/ois/real-time%20corba%20specification%2005-01-04%20jan%202005.pdf> [Accessed: 22/08/2013].
- [139] H. Oguma, A. Yoshioka, M. Nishikawa, R. Shigetomi, A. Otsuka, and H. Imai. New Attestation Based Security Architecture for In-Vehicle Communication. In *Proceedings of the 2008 IEE GLOBECOM Conference: GLOBECOM '08*, pages 1909–1914, 2008.
- [140] OPEN Alliance. OPEN Alliance Special Interest Group. Website. [Online]. Available: <http://www.opensig.org/> [Accessed: 22/08/2013].
- [141] Oracle. Java Native Interface. Java SE documentation, 2011. [Online]. Available: <http://docs.oracle.com/javase/6/docs/technotes/guides/jni/> [Accessed: 22/08/2013].
- [142] O. Organization. eXtensible Access Control Markup Language (XACML) Version 3.0. Technical report, OASIS Standards, 2013.

-
- [143] Oversee Project. Open Vehicular Secure Platform. Website. [Online]. Available: <https://www.oversee-project.com/index.php?id=2> [Accessed: 22/08/2013].
- [144] J. Pleumann. Really Fast Android: AMG Performance Media. Presentation slides (Droidcon '12), 2012. [Online]. Available: <http://de.slideshare.net/droidcon/really-fast-android> [Accessed: 22/08/2013].
- [145] G. Portokalidis, P. Homburg, K. Anagnostakis, and H. Bos. Paranoid Android: Versatile Protection for Smartphones. In *Proceedings of the 26th Annual Computer Security Applications Conference: ACSAC '10*, pages 347–356, New York, NY, USA, 2010. ACM.
- [146] G. Portokalidis, A. Slowinska, and H. Bos. Argos: an Emulator for Fingerprinting Zero-day Attacks for Advertised Honeypots with Automatic Signature Generation. In *Proceedings of the 1st ACM SIGOPS/EuroSys European Conference on Computer Systems 2006: EuroSys '06*, pages 15–27, New York, NY, USA, 2006. ACM.
- [147] J. Postel and J. Reynolds. File Transfer Protocol. RFC 959 (INTERNET STANDARD), Oct. 1985. Updated by RFCs 2228, 2640, 2773, 3659, 5797.
- [148] F. Qin, C. Wang, Z. Li, H.-s. Kim, Y. Zhou, and Y. Wu. LIFT: A Low-Overhead Practical Information Flow Tracking System for Detecting Security Attacks. In *Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture: MICRO '06*, pages 135–148, Washington, DC, USA, 2006. IEEE Computer Society.
- [149] A. Ramachandran, Y. Mundada, M. Tariq, and N. Feamster. Securing Enterprise Networks Using Traffic Tainting. *Special Interest Group on Data Communication*, 2008.
- [150] E. Rescorla and N. Modadugu. Datagram Transport Layer Security Version 1.2. RFC 6347 (Proposed Standard), Jan. 2012. [Online]. Available: <http://www.ietf.org/rfc/rfc6347.txt> [Accessed: 22/08/2013].
- [151] R. L. Rivest, A. Shamir, and L. Adleman. A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.
- [152] I. Rouf, R. Miller, H. Mustafa, T. Taylor, S. Oh, W. Xu, M. Gruteser, W. Trappe, and I. Seskar. Security and Privacy Vulnerabilities of In-car Wireless Networks: a Tire Pressure Monitoring System Case Study. In *Proceedings of the 19th USENIX conference on Security: USENIX Security '10*, pages 21–21, Berkeley, CA, USA, 2010. USENIX Association.

- [153] I. Roy, D. E. Porter, M. D. Bond, K. S. McKinley, and E. Witchel. Laminar: Practical Fine-grained Decentralized Information Flow Control. *SIGPLAN Notices*, 44(6):63–74, 2009.
- [154] F. Sagstetter, M. Lukasiewicz, S. Steinhorst, M. Wolf, A. Bouard, W. R. Harris, S. Jha, T. Peyrin, A. Poschmann, and S. Chakraborty. Security Challenges in Automotive Hardware/Software Architecture Design. In *Proceedings of the 16th International Conference on Design, Automation and Test in Europe: DATE '13*, pages 458–463, 2013.
- [155] K. Sampigethaya, R. Poovendran, and L. Bushnell. Security of Future E-enabled Aircraft Ad-hoc Network. White paper, 2008. [Online]. Available: <http://www.ee.washington.edu/research/nsl/papers/atio-08.pdf> [Accessed: 22/08/2013].
- [156] H. Schweppe and Y. Roudier. Security and Privacy for In-vehicle Networks. In *Proceedings of the 1st IEEE SECON International Workshop on Vehicular Communications, Sensing, and Computing: VCSC '12*, 2012.
- [157] P. Schönenberg. Introduction of Ethernet. Presentation slides (6th Vector Congress), 2012. [Online]. Available: http://www.vector.com/portal/medien/cmc/events/commercial_events/VectorCongress_2012/VeCo12_8_NewBusSystems_1_Schoenberg_Lecture.pdf [Accessed: 22/08/2013].
- [158] scut, team teso. Exploiting Format String Vulnerabilities. Technical Report version 1.2, Stanford Crypto Group, 2001.
- [159] SeVeCom project. Secure Vehicle Communications. Website. [Online]. Available: <http://www.sevecom.org/> [Accessed: 22/08/2013].
- [160] H. Shacham, M. Page, B. Pfaff, E.-J. Goh, N. Modadugu, and D. Boneh. On the Effectiveness of Address-space Randomization. In *Proceedings of the 11th ACM conference on Computer and communications security: CCS '04*, pages 298–307, New York, NY, USA, 2004. ACM.
- [161] V. Shankar, G. Urban, and F. Sultan. Online Trust: a Stakeholder Perspective, Concepts, Implications, and Future Directions. *Journal of Strategic Information Systems*, 11(3-4):325–344, 2002.
- [162] S. Shepler, M. Eisler, and D. Noveck. Network File System (NFS) Version 4 Minor Version 1 Protocol. RFC 5661 (Proposed Standard), Jan. 2010.
- [163] sim^{TD} project. Sichere Intelligente Mobilität Testfeld Deutschland. Website. [Online]. Available: <http://www.simtd.org/> [Accessed: 22/08/2013].

-
- [164] E. Slivka. Apple Pulls Russian SMS Spam App from App Store. Internet article, 2012. [Online]. Available: <http://www.macrumors.com/2012/07/05/apple-pulls-russian-sms-spam-app-from-app-store/> [Accessed: 22/08/2013].
- [165] Symantec. SSL Certificates from Symantec Powered by VeriSign. Website, 2013. [Online]. Available: <http://www.verisign.com/> [Accessed: 22/08/2013].
- [166] A. Tajeddine, A. Kayssi, and A. Chehab. A Privacy-Preserving Trust Model for VANETs. In *Proceedings of the 10th IEEE International Conference on Computer and Information Technology: CIT '10*, pages 832–837, 2010.
- [167] H. Teso. Aircraft Hacking - Practical Aero Series - In Materials of the 2013 Hack In The Box conference. Presentation slides, 2013. [Online]. Available: <http://conference.hitb.org/hitbsecconf2013ams/materials/D1T1%20-%20Hugo%20Teso%20-%20Aircraft%20Hacking%20-%20Practical%20Aero%20Series.pdf> [Accessed: 22/08/2013].
- [168] N. Thanthry, M. Ali, and R. Pendse. Security, Internet Connectivity and Aircraft Data Networks. In *Proceedings of the 39th International Carnahan Conference on Security Technology: CCST '05*, pages 251–255, 2005.
- [169] Trusted Computing Group. Trusted Platform Module (TPM) Summary. White paper. [Online]. Available: http://www.trustedcomputinggroup.org/resources/trusted_platform_module_tpm_summary [Accessed: 22/08/2013].
- [170] S. Tse and S. Zdancewic. Run-time Principals in Information-flow Type Systems. *ACM Transactions on Programming Languages and Systems*, 30(1), 2007.
- [171] US Department of Defense. Trusted Computer System Evaluation Criteria (Orange Book), 1983.
- [172] N. Vachharajani, M. J. Bridges, J. Chang, R. Rangan, G. Ottoni, J. A. Blome, G. A. Reis, M. Vachharajani, and D. I. August. RIFLE: an Architectural Framework for User-Centric Information-Flow Security. In *Proceedings of the 37th annual IEEE/ACM International Symposium on Microarchitecture: MICRO '04*, pages 243–254, Washington, DC, USA, 2004. IEEE Computer Society.
- [173] L. Völker and M. Schöller. Secure TLS: Preventing DoS Attacks with Lower Layer Authentication. In *Proceedings of the 15th Fachtagung Kommunikation in Verteilten Systemen: KiVS '07*, pages 237–248, 2007.
- [174] L. Walchshausl, R. Lindl, K. Vogel, and T. Tatschke. Detection of Road Users in Fused Sensor Data Streams for Collision Mitigation. In J. Valldorf and W. Gessner, editors, *Proceedings of the 10th International Forum on Advanced Microsystems*

- for Automotive Applications: AMAA '06*, pages 53–65, Berlin, Germany, 2006. VDI/VDE/IT.
- [175] D. Warne. Intel Atom-based car stereos coming. Internet article, 2009. [Online]. Available: <http://www.apcmag.com/intel-atom-based-car-stereos-coming.htm> [Accessed: 22/08/2013].
- [176] K. Weckemann. Herausforderungen in der Kommunikationsabstraktion (Middleware). Presentation slides (final event of the SEIS project), 2011. [Online]. Available: http://www.strategiekreis-elektromobilitaet.de/public/projekte/seis/das-sichere-ip-basierte-fahrzeuggbordnetz/pdfs/TP3_Vortrag1.pdf [Accessed: 22/08/2013].
- [177] K. Weckemann, D. Herrscher, A. Camek, P. C. G. Grotewold, oliver Hartkopp, P. Heinrich, C. Helmholz, M. Mandersheid, H. Meier, A. Kern, M. Kicherer, L. Völker, and M. Pfannenstein. SEIS AP 3.1: Grundlagen, Funktionsinteraktion und Migrationstrategie. Technical report, SEIS Project, 2011.
- [178] K. Weckemann, H.-T. Lim, and D. Herrscher. Practical Experiences on a Communication Middleware for IP-based In-car Networks. In *Proceedings of the 5th International Conference on COMmunication System softWARE and middlewaRE: COMSWARE '11*, page 12, 2011.
- [179] K. Weckemann, F. Satzger, L. Stolz, D. Herrscher, and C. Linnhoff-Popien. Lessons from a Minimal Middleware for IP-based In-car Communication. In *Proceedings of the IEEE Intelligent Vehicles Symposium: IVS '12*, pages 686–691, 2012.
- [180] B. Weyl, M. Graf, and A. Bouard. Smart Apps in einem vernetzten (auto)mobilen Umfeld: IT-Security und Privacy. In S. Verclas and C. Linnhoff-Popien, editors, *Smart Mobile Apps*, Xpert.press, pages 43–58. Springer Berlin Heidelberg, 2012.
- [181] B. Weyl, M. Wolf, F. Zweers, T. Gendrullis, M. S. Idrees, H. Schweppe, and Y. Roudier. D3.2: Secure On-board Architecture Specifications. Technical report, EVITA Project, 2010.
- [182] B. Weyl, M. Wolf, F. Zweers, T. Gendrullis, M. S. Idrees, H. Schweppe, and Y. Roudier. D3.2: Secure On-board Architecture Specifications. Technical report, EVITA Project, 2010.
- [183] J. White, B. Dougherty, R. E. Schantz, D. C. Schmidt, A. A. Porter, and A. Corsaro. R&D Challenges and Solutions for Highly Complex Distributed Systems: a Middleware Perspective. *Journal of Internet Services and Applications*, 3(1):5–13, 2012.

- [184] W. Wiewesiek. SHE - Data Security for Automotive Embedded Systems. Presentation slides (Workshop on Cryptography and Embedded Security Embedded World), 2012. [Online]. Available: https://www.escript.com/fileadmin/escript/pdf/WEB_Secure_Hardware_Extension_Wiewesiek.pdf [Accessed: 22/08/2013].
- [185] N. Williams. Internet Draft – IPsec Channels: Connection Latching, draft-ietf-btnc-connection-latching-00.txt. Technical report, Internet Engineering Task Force (IETF), 2009.
- [186] M. Wolf, A. Weimerskirch, C. Paar, and M. Bluetooth. Security in Automotive Bus Systems. In *Proceedings of the Workshop on Embedded Security in Cars: ESCAR '04*, 2004.
- [187] L. Xie, X. Zhang, J.-P. Seifert, and S. Zhu. pBMDS: a Behavior-based Malware Detection System for Cellphone Devices. In *Proceedings of the 3rd ACM conference on Wireless network security: WiSec '10*, pages 37–48, New York, NY, USA, 2010. ACM.
- [188] H. Yin, D. Song, M. Egele, C. Kruegel, and E. Kirda. Panorama: Capturing System-wide Information Flow for Malware Detection and Analysis. In *Proceedings of the 14th ACM conference on Computer and communications security: CCS '07*, pages 116–127, New York, NY, USA, 2007. ACM.
- [189] A. Zavou, G. Portokalidis, and A. D. Keromytis. Taint-exchange: a Generic System for Cross-process and Cross-host Taint Tracking. In *Proceedings of the 6th International workshop on Advances in information and computer security: IWSEC '11*, pages 113–128, Berlin, Heidelberg, 2011. Springer-Verlag.
- [190] N. Zeldovich, S. Boyd-Wickizer, E. Kohler, and D. Mazières. Making Information Flow Explicit in HiStar. In *Proceedings of the 7th USENIX Symposium on Operating Systems Design and Implementation: OSDI '06*, pages 19–19, Berkeley, CA, USA, 2006. USENIX Association.
- [191] N. Zeldovich, S. Boyd-Wickizer, and D. Mazières. Securing Distributed Systems with Information Flow Control. In *Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation: NSDI'08*, pages 293–308, Berkeley, CA, USA, 2008. USENIX Association.
- [192] Q. Zhang, J. McCullough, J. Ma, N. Schear, M. Vrable, A. Vahdat, A. C. Snoeren, G. M. Voelker, and S. Savage. Neon: System Support for Derived Data Management. In *Proceedings of the 6th ACM SIGPLAN/SIGOPS conference on Virtual Execution Environments: VEE '10*, pages 63–74, 2010.

A Appendix

A.1 Numerical values of Figure 5.5

Buffer size (bit)	Throughput (call/sec)					Bandwidth (kbit/sec)				
	(1)	(2)	(3)	(4)	(5)	(1bis)	(2bis)	(3bis)	(4bis)	(5bis)
32	1987	1093	931	767	633	509	280	238	196	162
64	1984	1057	948	682	635	1016	541	485	349	325
128	1987	996	935	665	640	2035	1020	957	671	655
256	1865	920	706	674	479	3820	1884	1446	1380	981
512	1651	778	695	473	430	6762	3187	2847	1937	1761

A.2 Numerical values of Figure 5.6

Buffer size (bit)	Throughput (call/sec)				Bandwidth (kbit/sec)			
	(1)	(2)	(3)	(4)	(1bis)	(2bis)	(3bis)	(4bis)
32	914	520	507	395	234	133	130	101
64	865	487	468	363	443	250	240	186
128	777	442	426	328	795	452	437	336
256	776	438	424	329	1589	897	868	674
512	700	392	377	300	2867	1607	1543	1229

A.3 Numerical values of Figure 5.7

Buffer size (bit)	Throughput (call/sec)			Bandwidth (kbit/sec)		
	(1)	(2)	(3)	(1bis)	(2bis)	(3bis)
32	1201	639	525	308	163	134
64	1163	643	537	595	329	275
128	1114	642	525	1141	657	537
256	1073	629	519	2197	1289	1063
512	986	625	523	4039	2560	2142

Curriculum Vitae



Personal Details

Alexandre Bouard
born April 20th, 1987 in Clamart, France

PhD

10/2010 – 09/2013 BMW Research and Technology, Munich, Germany
Technische Universität München (TUM), Munich, Germany
IT Security Group,
Prof. Dr. Claudia Eckert

Topic “Middleware-based Security for Future In-Car Networks”

Studies

09/2007 – 09/2010 TELECOM ParisTech, Paris, France
Master’s Degree in Engineering
- Specialization Area: Computer Science
- Master Thesis: “Development of an Automotive Security
Middleware, Etch”

09/2008 – 09/2010 EURECOM, Sophia Antipolis, France
Research Institute related to TELECOM ParisTech
- Specialization Area: Communication System Security

09/2005 – 07/2007 Classes Préparatoires in lycée du Parc, Lyon, France
- Specialization Area: Mathematics, Physics, Chemistry

Experience

- 11/2013 – now BMW AG, Munich, Germany
- Project Manager/Electromobility domain
- 10/2010 – 09/2013 BMW Research and Technology, Munich, Germany
- Internship, Software development
- Master thesis

Miscellaneous

- Education Lycée Saint Exupéry, Lyon, France (1998 - 2005)
- Languages French (native language)
 English (business fluent)
 German (business fluent)

München, August 29, 2014