Security and Privacy Aspects of Automotive Systems

Submitted by

Hafizah Mansor

for the degree of Doctor of Philosophy

of the

Royal Holloway, University of London

2017

Declaration

This doctoral study was conducted under the supervision of Professor Konstantinos Markantonakis.

The work presented in this thesis is the result of original research carried out by myself, in collaboration with others, whilst enrolled in the Department of Mathematics as a candidate for the degree of Doctor of Philosophy. This work has not been submitted for any other degree or award in any other university or educational establishment.

Signed ....[illegible]....... (Hafizah Mansor)
Date: 19/07/2017

Publications

A number of papers from this work have been published and presented at international conferences:

- H. Mansor, K. Markantonakis, and K. Mayes, “CAN bus risk analysis revisit,” in Information Security Theory and Practice, Securing the Internet of Things, (Heraklion, Greece), pp. 170–179, Springer, Jun 30-July 2 2014.
- H. Mansor, K. Markantonakis, R. N. Akram, and K. Mayes, “Don’t brick your car: Firmware confidentiality and rollback for vehicles,” in Availability, Reliability and Security (ARES), 2015 10th International Conference on, (Toulouse, France), pp. 139–148, IEEE, Aug 24-27 2015.
- H. Mansor, K. Markantonakis, R. N. Akram, and K. Mayes, “Let’s get mobile: Secure FOTA for automotive system,” in Network and System Security, (New York, USA), pp. 503–510, Springer, Nov 3-5 2015.
- H. Mansor, K. Markantonakis, R. N. Akram, K. Mayes, and I. Gurulian, “Log your car: The non-invasive vehicle forensics,” in Trust, Security and Privacy in Computing and Communications (TrustCom), 2016 IEEE 15th International Conference on, (Tianjin, China), IEEE, Aug 23-26 2016.
- H. Mansor, K. Markantonakis, R. N. Akram, K. Mayes, and I. Gurulian, “Log your car: Reliable maintenance services record,” in Information Security and Cryptology (Inscrypt), 12th International Conference on, (Beijing, China), Springer, Nov 4-6 2016.

Acknowledgement

I would like to acknowledge and thank the many people who have helped me to bring this thesis to completion:

Firstly, my thanks go to my supervisor, Prof. Konstantinos Markantonakis. His support and guidance have helped me tremendously in this work. His patience and advice while guiding me in completing my PhD really inspired me. My thanks and appreciation also go to Raja Naeem Akram, whose advice, support and guidance were my biggest motivations while completing the work, and to Prof. Keith Mayes, whose insightful guidance has always brought me inspiration in my work. The three of you are my inspiration to be a successful researcher.

Special thanks to my colleague, Iakovos Gurulian, who helped me in the implementation of Android applications. Also to all my colleagues at the Smart Card Centre, especially to Sheila Cobourne who reviewed my work so patiently. Thanks to all others for the ideas and knowledge sharing sessions during our weekly meetings. I am blessed to have all of you as my colleagues.

My appreciation goes to the Ministry of Higher Education Malaysia and the International Islamic University of Malaysia (IIUM) for sponsoring my study at Royal Holloway University of London, which has been a life-changing experience.

No words can describe my gratitude to my husband, Mohd Zaki Mohtar and my two wonderful daughters, Khaira and Naura, for their amazing support in being understanding all throughout my PhD journey. Your patience and smiles always boosted my spirits and made me realize how blessed I am. My gratitude and appreciation to my family, especially my parents, my parents-in-law and my eldest sister, Hazmin Mansor, for their prayers and support. Our weekly Skype conversations were my motivational boosters. My thanks also go to my wonderful friends in Egham for the great gatherings and delicious food. Without all of you, this journey would not have been possible.

Abstract

Connected cars have Internet access and connectivity to supporting infrastructures and other vehicles. This enables them to take advantage of smart applications (such as crash alert and real-time location) that can be used in different phases of the car's life cycle. Automotive systems consist of a number of subsystems and networks that operate a car. In this thesis, we analyse the security and privacy aspects of automotive systems in connected cars. The analysis is performed through selected vehicular applications; the vehicular firmware update, forensics and maintenance logging systems. The applications are selected to cover various aspects of security during the different phases of the car's life cycle. For each vehicular application, the process, challenges and requirements are considered. Our analysis of the security and privacy requirements of the automotive systems provides valuable insights into the overall reliability and safety of connected cars.

Firstly, we looked into the firmware update application for vehicular systems as this is crucial in ensuring the safety and reliability of the car. One of the main industrial projects, the E-safety Vehicle Intrusion Protected Applications (EVITA) project, has proposed a firmware update over-the-air protocol. We found some shortcomings in the operations of this protocol and proposed an improvement, EVITA+, which provides additional assurance by considering both security and general requirements to ensure a successful update. Secondly, we propose a firmware update protocol using a mobile device, with a mechanism that protects the intellectual property of the firmware and ensures reliability during updates. Thirdly, in the forensics application, the main consideration of the protocol is to protect data privacy while giving access to the car owner. Finally, in the maintenance logging application, we give data access to the car owner, while ensuring authenticity. The exploitation of mobile devices in the proposed applications provides user flexibility as well as privacy protection, and differentiates our solutions from current industrial implementations.

The proposed protocols are analysed using automated formal analysis tools, i.e. CasperFDR and Scyther, and implemented to prove feasibility and determine performance.

Contents

List of Figures

List of Tables

List of Abbreviations

ABS	Anti-locking Brake System
ADAS	Advanced Driver Assistance Systems
AES	Advanced Encryption Standard
ASIC	Application Specific Integrated Circuit
CAN	Controller Area Network
CCU	Central Communication Unit
CSP	Communication Sequential Process
DoS	Denial of Service
DRM	Digital Rights Management
DT	Diagnostic Tool
ECC	Elliptic Curve Cryptography
ECU	Electronic Control Unit
EDR	Event Data Recorder
EEPROM	Electrically Erasable Programmable Read-Only Memory
EVITA	E-Safety Vehicle Intrusion Protected Applications
FDR	Failure Divergence Refinement
FOTA	Firmware update Over-The-Air
GUI	Graphical User Interface
HSM	Hardware Security Module
IP	Intellectual Property
JTAG	Joint Test Action Group
LIN	Local Interconnect Network
MAC	Message Authentication Code
MOST	Media Oriented Systems Transport
NVM	Non-Volatile Memory
OBD	On-Board Diagnostic
OEM	Original Equipment Manufacturer
OTA	Over-The-Air
PII	Personally Identifiable Information
RAM	Random Access Memory
RKE	Remote Keyless Entry
RNG	Random Number Generator
ROM	Read-Only Memory
RSA	Rivest Shamir Adleman
RTE	Runtime Environment
SHA	Secure Hash Algorithms
SHE	Secure Hardware Extension
TCB	Trusted Computing Base
TCG	Trusted Computing Group
TPM	Trusted Platform Module
TPMS	Tire Pressure Monitoring Systems
UTC	Coordinated Universal Time
V2I	Vehicle-to-Infrastructure
V2V	Vehicle-to-Vehicle
V2X	Vehicle-to-X
VERONICA	Vehicle Event Recording based On Intelligent Crash Assessment
VIN	Vehicle Identification Number

Chapter 1

Introduction

Contents

Connected cars are cars that have communications capability; typically Internet access. This allows them to be connected with supporting infrastructure and other connected vehicles. We begin the thesis with a brief introduction to connected cars. The general motivations in exploring security aspects in automotive systems, together with the challenges of connected cars, are discussed. Next, our contributions to improve the security and privacy aspects of the automotive systems are briefly described. Finally, we outline the structure of the thesis.

1.1 Brief Introduction of Connected Cars

Connectivity provides the leverage for a wide range of applications accessible via the car. The term connected vehicles refers to applications, services and technologies that connect a vehicle to its surroundings [1]. Connected cars are different from traditional cars in terms of their accessibility and communication with the outside world. Many smart applications are associated with connected cars, providing a wide range of appealing features to car owners/drivers, especially in terms of safety applications. For example, applications such as Advanced Driving Assistance Systems (ADAS) [2] could improve driving performance, and thus decrease the possibility of accidents. Another prominent application is the eCall system [3] which has been introduced to provide emergency assistance. This will be compulsory in all new cars in Europe by 2018 [4].

In order for these applications to work, the car communicates with other vehicles through Vehicle-to-Vehicle (V2V) communication, or with smart infrastructure through Vehicle-to-Infrastructure (V2I) communication. The connectivity of a car to an external entity (infrastructure, or another vehicle) is currently through IEEE 802.11p [5] and cellular networks [1]. The car should be equipped with smart sensors and controllers to be able to support the connected car features.

Car manufacturers are competing to provide the most appealing and the best features for connected cars [6, 7, 8, 9]. Generally, most users are not aware of the security implications of these applications, including the privacy impact [10]. Security researchers have shown how vulnerable the existing systems are [11, 12, 13, 14]. A more detailed discussion of connected cars will be presented in the next chapter.

1.1.1 Definition of Terms Within the Scope of the Thesis

Security is required for automotive systems. This includes security for the in-vehicle nodes and communications, security for the car's external communication to supporting infrastructure or other vehicles, and security with the actors in the overall ecosystems. Security issues in automotive systems can impact the reliability and safety of the systems.

Automotive systems are divided into safety-critical and non-safety-related systems. If any of the safety-critical systems are compromised, this could cause safety issues such as controlling a car by braking remotely [15]; whereas compromise of the entertainment system would not have a direct safety impact.

Reliability in automotive systems depends on the underlying systems performing their required functions in all conditions. A reliable car usually means less maintenance or repair is required. In automotive systems, reliability may impact safety; for example, a sudden breakdown may cause an accident. In automotive forensics, reliability implies availability of the data. A reliable forensic system would require the data to be available for forensic investigation.

Privacy is the ability of a group or individual to seclude themselves, and to choose to express/share selectively. In this context, the privacy of the car owner is her/his ability to choose what data to share and with whom.

Confidentiality is the concealment of data from the ecosystem; only authorised parties are permitted to access protected data.

Flexibility is the ease of use and user-friendliness of the applications in providing data access and privacy protection to the car owner.

1.2 Motivations and Challenges

With the introduction of connected cars, security is seen as a critical aspect in ensuring a safe and reliable system [12, 16]. Although smart applications are very beneficial to the whole automotive ecosystem, especially car users, there are many vulnerabilities associated with their introduction [17, 18].

In automotive systems, one of the main concerns is the reliability of the system [19]. Reliability is related to correctness of the data, and to safety. For example, for updates or logging, the integrity of the stored data is vital; otherwise this could adversely affect the safety of the car's operations.

It is also a challenge to ensure the authenticity of the entities involved in a particular communication since the automotive ecosystem contains a number of stakeholders (with different interests), which will be discussed in the next chapter.

Confidentiality of data is another aspect to consider as it affects privacy, intellectual property protection and safety.

Regardless of the stakeholders and their different interests, availability of data must be ensured so that the system operates accordingly.

Data ownership is a challenge in many systems and automotive applications are no exception. Having multiple stakeholders complicates this challenge; for example, the car owner could possibly own data, but may not have access to it.

1.3 Contributions

In this thesis, we analyse the security and privacy aspects of automotive systems and connected cars, which then relate to the safety and reliability of the cars. The automotive industry has now realised the importance of security and is moving towards using security-enabled devices [20, 21, 22, 23]. Our work analyses the security of vehicular applications in the existing implementations of the automotive industry and proposes improvements to enhance security.

The analysis of security and privacy in automotive systems is performed through three different vehicular applications: firmware updates, forensics, and maintenance logging systems. In the first study, we considered one of the main industrial projects in the automotive field; the E-safety Vehicle Intrusion Protected Applications (EVITA) project [21]. We found some shortcomings in the operation of EVITA for firmware updates.

Using the concept of the EVITA platform, we investigated integrating a mobile device with automotive applications to provide flexibility for users. However, our overriding concern is to ensure that security and privacy are not compromised or violated. We propose three protocols for the three applications mentioned above. These proposed protocols use new architectures that provide flexibility to the users while at the same time protecting their privacy and ensuring the systems' security.

For firmware updates, while providing a user-friendly architecture, the intellectual property of the firmware needs to be protected. The firmware intellectual property copyright is held by the car manufacturer and is valuable to their business, especially when they are providing a special feature for their brand. Furthermore, the reliability of the updated firmware needs to be protected to ensure the safety of the overall systems. Our proposed protocol considers all these requirements.

In vehicular forensics, while protecting the privacy of the user, the user himself/herself may need access to the data. However, access to the data is either limited or not possible in current implementations. Our proposed protocol provides flexibility and data access to the user, and also considers the security aspects of the overall systems, especially in protecting the integrity of the data.

In the maintenance logging system, we want to make sure that users are given flexibility and data access. At the same time, we want to ensure that the data is correct and authentic. Using a mobile application, an automated maintenance logging system is proposed.

The proposed protocols were formally analysed using CasperFDR and Scyther tools and no known attacks have been found. The protocols were also implemented to measure their performance, and thus show their feasibility.

1.4 Thesis Outline

This thesis is organised as follows:

Chapter 2 and Chapter 3 introduce automotive systems and the three selected vehicular applications. Chapter 2 introduces automotive systems, in-vehicle communications, automotive ecosystems and automotive security. Next, in Chapter 3, three vehicular applications are described: firmware updates of the ECUs, vehicular forensics, and the maintenance services logging system. For each vehicular application, the process, challenges and requirements are discussed.

In Chapter 4, we discuss firmware updates for vehicular systems. We found some shortcomings in the firmware update over-the-air protocol proposed in the EVITA project. We then proposed an improved protocol, EVITA+, which provides additional assurance by considering both security and general requirements to ensure a successful update. These features can help to provide assurance on the reliability and safety of the car.

Chapter 5 provides another contribution on vehicular firmware updates. Security is the main consideration in the EVITA protocol, with limited reliability and flexibility considerations in the update process. Following on from our proposal to improve the EVITA protocol's reliability with regard to the update process, Chapter 5 looks into providing flexibility in terms of the user-friendliness of the update process. We consider the automotive components under careful control of the car manufacturer, especially the firmware update process, to ensure a reliable, safe and secure car. At the same time, we want to make sure the distribution of the firmware updates is flexible and consumer-friendly. This chapter proposes a secure firmware update protocol for automotive systems, using a mobile device.

Building on our idea of using a mobile device for automotive applications, Chapter 6 discusses a solution that uses a mobile device in vehicular forensics. Using a mobile application as a logging platform for vehicle operation may improve the effectiveness of forensic applications. Accuracy of forensic analysis can be further improved with a wider range of parameters of the collected data. While ensuring secure data and user privacy, a secure framework for vehicle forensics is proposed.

Chapter 7 describes our final contribution. A mobile application may provide an automated logging system for car maintenance services. A secure protocol is proposed to perform automated logging that can ensure the integrity of the records. The use of a mobile device provides a user interface as well as connectivity for the car. Finally in Chapter 8, we summarise all the contributions of the thesis.

Part I

Background

Chapter 2

Connected Cars

Contents

This chapter introduces connected cars with a discussion of automotive systems and in-vehicle communications. The actors in the vehicular ecosystem, who are the stakeholders that play different roles in the vehicular systems and applications, are also discussed. Automotive security, which includes the security challenges in automotive systems, is briefly explained.

2.1 Introduction

Automotive systems have migrated from mechanically controlled to digitally controlled systems. A smart car requires automation and connectivity. This introduces autonomous vehicle applications and connected cars. The term “connected cars” refers to applications, services and technologies that connect cars to their surroundings [1]. These applications work with either vehicle-to-vehicle or vehicle-to-infrastructure communications. An example of such applications is the Advanced Driver Assistance Systems (ADAS) [24].

Automotive security is of great importance as it is critically related to the safety and reliability of the vehicle. This includes the use of security modules and security mechanisms to protect the security of the car. Security mechanisms include hardware and software protection and secure communication within the vehicle [25]. The nodes, networks and applications of automotive systems are introduced in this chapter to aid understanding. A number of stakeholders in the automotive ecosystem have direct and indirect relationships with the car (this will be discussed in Section 2.4). Each of the stakeholders has different capabilities and intentions. The introduction of smart applications in connected cars introduces a number of security issues. While prohibiting cars from conducting external communications can reduce the attack vectors, such smart applications are still useful, and they require the car to be connected to an external entity such as diagnostic tools at workshops, and mobile devices.

Currently, car manufacturers, transport and road administrators, semiconductor manufacturers, software design houses and even payment systems collaborate to provide connected cars’ applications [26].

Chapter organisation Section 2.2 discusses the nodes in automotive embedded systems, i.e. the applications, requirements and the security platform. Next, Section 2.3 describes in-vehicle networks and their security aspects. The stakeholders in automotive ecosystems and the trust model are discussed in Section 2.4. We then summarise the general issues in connected cars in Section 2.5. The use of mobile devices and issues related to their use are discussed in Section 2.6.

2.2 Electronic Control Unit

Electronic Control Units (ECUs) are microcontrollers that control the overall operation of a car. In modern cars, depending on whether they are basic or luxury cars, there are about seventy ECUs that operate the car’s functionalities [21]. Each ECU has different functions and controls. These are divided into different types of operations;

for example, for body control, engine control and telematics. Signals from sensors are sent to the ECUs and the ECUs act accordingly as programmed and send appropriate instructions to the actuators to carry out the operations. Communications between ECUs are sent through different in-vehicle networks (to be discussed in Section 2.3) depending on the application of the ECU. These ECUs are interconnected with each other to perform different (collaborative) tasks. Therefore, if one or more ECUs are not functioning properly, they could be detrimental to the overall operations, safety and reliability of the vehicle.

2.2.1 Applications

An automotive system has four main subsystems: power train and engine; chassis and safety; body control; and multimedia and entertainment. Power train subsystems include engine management and transmission systems. The chassis and safety subsystems include the Anti-locking Brake System (ABS), power steering and braking systems. Body control subsystems are divided into comfort and safety. Comfort systems include heating and air conditioning, central locking, seat and mirror adjustment; while safety systems include airbag and restraint systems. Multimedia and entertainment subsystems include radio and phone systems [27]. These subsystems are now being cooperatively used for autonomous driving applications such as forward collision warning or braking and lane-keeping aids [28].

2.2.2 ECU Requirements

The ECU requirements are divided into hardware and software requirements. A cooperation between car manufacturers, suppliers, electronics and semiconductor providers and software providers established Automotive Open System Architecture (AUTOSAR). AUTOSAR provides a standardised open software architecture for automotive ECUs [29]. The three layers of software architecture are the application layer, the Runtime Environment (RTE), and the basic software layer. The basic software layer includes standardised software modules for drivers and communication modules, and the microcontroller abstraction. For the hardware, the specifications stipulate that a set of requirements must be met: secure storage, tamper resistance and a customised memory size depending on the application of the ECU. To ensure secure application, the use of a security module is required. The security module requires secure creation (which makes it unclonable in terms of the key and content to prevent illegal copying), secure integration, secure operation, secure storage, and a secure output channel [30].

2.2.3 ECU Security Module Candidates

Table 2.1: Features of different security modules

Security features	HIS-SHE	EVITA HSM			TPM	
		Full	Medium	Light	Rich	Thin
Algorithms						
ECC	○	●	○	○	●	●
RSA	○	●	○	○	●	●
AES	●	●	●	●	●	●
CMAC/HMAC	●	●	●	●	●	●
SHA-1	○	●	●	○	●	○
SHA-256	○	○	○	○	●	●
WHIRLPOOL	○	●	●	○	○	○
RNG	PRNG	TRNG	TRNG	PRNG	TRNG	TRNG
UTC Clock	○	●	●	●	○	○
Counter	○	●	●	○	●	●
Internal NVM	●	●	●	◐	○	○

Note: ○:Not supported, ◐:Optional, ●:Supported

There are three different types of security modules available for automotive applications. Table 2.1 shows the different security modules and their features [31, 32, 33]. The comparison criteria include the different algorithms supported, and the availability of a Random Number Generator (RNG) [34], Coordinated Universal Time (UTC) clock, replay protection monotonic counter and Non-Volatile Memory (NVM) on the different platforms. Algorithms supported are Elliptic Curve Cryptography (ECC) [35], Rivest Shamir Adleman (RSA) [36], Advanced Encryption Standard (AES) [37], Cipher-based Message Authentication Code (CMAC) [38], Hash-based Message Authentication Code (HMAC) [39], Secure Hash Algorithms (SHA-1, SHA-256) [40] and WHIRLPOOL (AES-based hash function) [41]. The three security modules are discussed as follows:

HIS-SHE Secure Hardware Extension (SHE) was introduced by Hersteller Initiative Software (HIS) in 2009 [33]. The idea is to add a secure zone to the ECU, where computation and data are protected. It has secure storage for cryptographic data and hardware cryptographic engines. Examples of available ECUs with SHE include microcontrollers by Infineon (TC1798 [42]) and Freescale (MPC5646 [43]). It was the first technology to consider security zones for ECUs in automotive systems, and only limited security algorithms were supported.

EVITA Hardware Security Module (HSM) The EVITA project (2008-2011) [21] proposed an architecture using Hardware Security Modules (HSMs). A range of HSM levels are used for different security functions required by the various ECU modules [44]. There are three types of HSMs: full HSM, medium HSM and light HSM. Full HSM is used in the Central Communication Unit (CCU) module, which is the ECU module responsible for Vehicle-to-X (V2X), and can be either Vehicle-to-Infrastructure (V2I) or Vehicle-to-Vehicle (V2V) communications. Medium HSMs are used for advanced ECUs (gateways, head unit, engine control) in in-vehicle networks (see Section 2.3). Light HSMs are used in sensors and actuators.

The EVITA project proposed an integrated HSM within each ECU. The proposed architecture was implemented in FPGA for proof of concept. At the time of writing, only medium and light HSM-based ECUs are available in the market in Application Specific Integrated Circuit (ASIC) solutions. A full HSM requires a high performance solution (32-bit CPU core, with crypto accelerators) to be able to meet the automotive requirements [**?**]. Examples of available ECUs with HSM include microcontrollers by Infineon (TC297 [45]) and Freescale (MPC5748 [46]).

Trusted Platform Module (TPM) The Trusted Computing Group (TCG) proposed the Automotive Thin and Rich TPM for automotive application in 2015 [47]. The Auto TPM is not integrated within the ECU. Each ECU is connected to an Automotive TPM. The Automotive TPM provides platform attestation and integrity reporting of software [48]. The Automotive Rich TPM is similar to the Full EVITA HSM, where it is used for the Head Unit or Central Communication Unit. Other nodes use Automotive Thin. Automotive Rich has the conceptual specification to be the same specification as TPM 2.0. To date, only Automotive Thin TPM specifications [32] are available from TCG. TCG proposes two different architectures. In the first architecture, only Automotive Thin TPMs are used for all the ECUs, including the CCU. In the second architecture, the Automotive Rich TPM is used for the CCU and Automotive Thin TPMs are used for all other ECUs. For Automotive Thin, the supported algorithms are RSA2048 or ECC256, AES128, SHA256 and HMAC.

2.3 In-vehicle Communications

In-vehicle communications involve communications between the ECUs via the in-vehicle networks.

2.3.1 Central Communication Unit (CCU)

The Central Communication Unit (CCU), also known as the head unit, is the central unit for all the ECUs in the in-vehicle networks. It is the first unit any external device will need to communicate with in order to talk to other ECUs within the vehicle. It stores all the identifications (ID), public keys (pk) and symmetric keys (k) for all the ECUs.

2.3.2 In-vehicle Networks

There are a number of in-vehicle networks, which include the Controller Area Network (CAN), Local Interconnect Network (LIN) [49], Media Oriented Systems Transport (MOST) [50] and FlexRay [51]. Different networks are used for different applications as shown in Table 2.2 [52].

All the in-vehicle networks (other than CAN) are interconnected through the CAN bus. Since the CAN bus is the main in-vehicle communication network used in automotive applications [21], our discussion will focus only on the CAN bus. A CAN bus is a broadcast serial communication protocol where every node on the bus can transmit and receive messages to and from the bus. In a car, the nodes are the ECUs. The CAN bus can be a two-wire or single wire cable with all the nodes attached and a terminating node with a 120 Ohm resistor at each end. It was first introduced by Bosch. The CAN physical signaling and data link layer protocols are standardised in ISO 11898-1 [53]. It is configurable to standard frame format (supports 11 bit length ID message) or extended frame format (supports 29 bit length ID message). There are four types of frame: data frame, remote frame, error frame and overload frame. It is a priority-based message protocol where the message with higher priority gets transmitted first. The priority of the message depends on the arbitration bit (the ID of the message). A lower value of ID will win the arbitration and be transmitted first. The message with the higher ID value will wait its turn. There are 2 different ranges of CAN frequencies: high frequency CAN, which operates up to 1 Mbps and low frequency CAN, which operates from 40 kbps up to 125 kbps.

For each network, there are local gateways. For the different networks to communicate, there is a node called the super gateway [52]. The super gateway operates on a CAN bus network. As shown in Figure 2.1, the central ECU is called the Central Communication Unit (CCU). This unit acts as a central node. It is accessible to allow external devices to communicate with the in-vehicle ECUs.

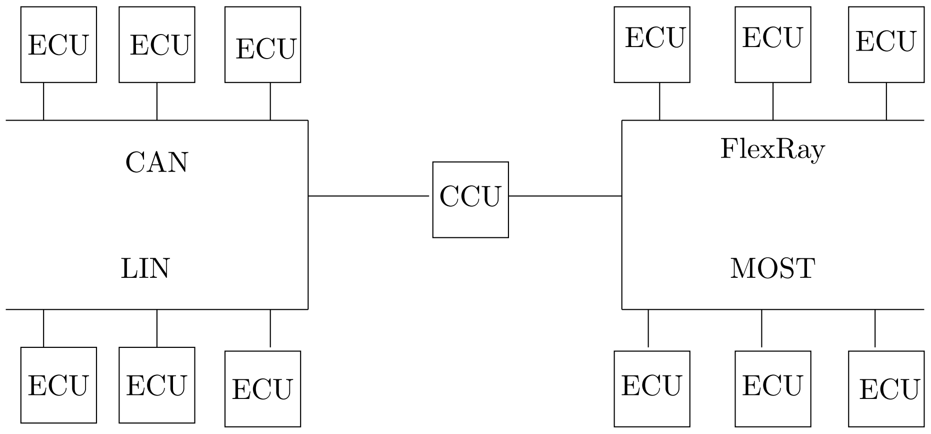

Figure 2.1: In-vehicle networks

Table 2.2: In-vehicle networks

Network	Maximum data rate	Applications
CAN	1 Mbps	Power train and engine, Chassis and safety
LIN	20 kbps	Body control
MOST	24 Mbps	Multimedia and entertainment
FlexRay	10 Mbps	Drive-by-X

2.3.3 Threats and Vulnerabilities

As shown in many studies (to be discussed later in this section), connected cars can be attacked through physical and remote access. The vulnerabilities lie in the network architecture and protocols, the ECU itself and the cyber physical applications [54]. Physical access gives a wide range of attack surfaces, especially if the attacker is able to get access to the ECU(s) and/or network. Indirect physical access includes external interfaces such as compact disc (CD) and On-Board Diagnostic (OBD) port, while remote access is through wireless communication, whether short or long range [12, 13].

A modification attack is possible by modifying the content on the CAN bus; i.e. by injecting unauthorised messages. For example, in [16] the researchers were able to change the display on the instrument panel cluster. A masquerading attack is possible by faking the identity of an authorised device to gain access or to communicate with the in-vehicle network; for example, masquerading as an authorised diagnostic tool. A man-in-the-middle attack could be performed using a piggy-back device on an authorised ECU, and relaying and/or altering the content of messages in the network. An eavesdropping attack is also possible by putting an additional ECU on the CAN network to tap/listen to the CAN network. The attacker is then able to listen to all the messages flowing through the network since it is a broadcast network where all the nodes on the network can listen to the messages. A replay attack is possible once an attacker is able to eavesdrop and learn from the messages (and maybe reverse-engineer them to understand what each message does). The attacker can then reuse the messages to replay them. A DoS attack is also possible on a CAN network. Since CAN is serviced based on the priority of its ID, an attacker can keep on sending messages with a high priority to congest the network and deny service for other lower priority messages.

Potential attacks on vehicles were highlighted in 2010 when security researchers proved that by injecting messages to the CAN bus, they were able to control a car in several ways, including controlling the display on the radio and the speedometer, and killing the engine [16]. The most prominent attacks on automotive systems were performed by Charlie Miller and Chris Valasek. In 2013, based on other work that showed the possibility of remote code execution via interfaces such as Bluetooth and telematics units [16], they demonstrated the extension attacks that become possible following a remote code execution attack. By using two car models, they produced tools to conduct the attacks. They showed the different CAN messages used to control the different operations of the cars in the attack [11]. They were able to control the steering, braking, acceleration and displays on the cars. In 2014, they analysed the possibility of performing remote attacks on 20 car models. They quantified the difficulty of performing the attack through remote attack surfaces, cyber physical features and the car's network architecture. They also proposed a number of defence mechanisms, including secure remote endpoints, secure in-vehicle network, authentic non-compromised parts and attack detection [54]. In 2015, they performed attacks on a Cherokee jeep [55], as it had been found to be the most vulnerable car based on their previous work [54]. They showed that it is possible to conduct an attack on the car without physical access, and affect operations such as killing the engine and controlling the steering. They implemented the attacks as proposed by [12] and documented each step and the corresponding message/input file in performing the attacks. The attacks were performed on almost all possible features of attack surfaces, including the telematics system that provides connection to a cellular network. After they had gained access remotely, they sent malicious CAN messages and were able to control the car.

Attacks can be conducted using the vulnerabilities of network architecture and protocols, ECUs and cyber physical applications.

Network architecture and protocols: Since CAN bus operations involve reception and transmission of messages through the CAN bus network, the risks are analysed from the weaknesses in these operations that might be used by an attacker to cause failure modes. For example, for message reception, an attacker can cause failure by making improper filtering of messages: under-filtering will cause acceptance of unwanted or unnecessary messages, and over-filtering will cause loss of messages. Another weakness of the CAN bus is the broadcast nature of the network, where every node on the network can see the message. Limitation of the number of data bytes per message transmission would cause performance issues when security data is to be included in the message transmission. Priority bus-based arbitration can be manipulated by sending higher priority messages to cause denial of service. Last but not least, there is an unlimited number of nodes that can be connected to the CAN bus, with minimal or no authorisation required. Manipulations through these weaknesses can cause severe effects if they involve critical ECUs such as the electronic brake control module. The CAN bus can be attacked in many ways, including the exposure of the CAN bus through the OBD port [56], and during the ECU's firmware update [30]. If an attacker is able to access the CAN bus, he/she can sniff the messages, and later use reverse engineering techniques to decode the messages. This could cause loss of intellectual property if the firmware or the operations are obtained. If sensitive data is leaked or obtained, it could generate privacy and confidentiality issues. If denial of service attacks and/or integrity manipulation attacks are performed, the safety/reliability of the systems could be affected.

ECU: Issues such as malware, which are common in the computer world, are now emerging in automotive systems. In order to have access to ECUs, attackers are no longer required to have direct physical access. They can get to an ECU through the networks or even by remote access. The ECU can be attacked through a weakness in design and implementation of its hardware, software or applications. Malware can be injected through OBD ports, firmware updates and removable media ports [57].

Cyber physical applications: Tire Pressure Monitoring Systems (TPMS) that use radio frequency signals to monitor the condition of the car tyres can be easily attacked through eavesdropping and spoofing [17]. Such attacks can also cause privacy issues as the car can be tracked. In order to solve the TPMS issues, Solomon *et al.* proposed a solution for securing the TPMS using lightweight crypto such as SPECK and PRESENT [58].

The Remote Keyless Entry (RKE) system for vehicle access is vulnerable to many attacks [59]. The KeeLoq block cipher, which has been widely used for RKE systems, has been found to have many vulnerabilities [60, 61]. A relay attack was demonstrated to prove the vulnerability of the RKE system [62]. Vehicle immobilisers have also been subjected to attacks. Implementation issues using proprietary algorithms such as

Hitag2 were also proven to be an attack surface [63]. By eavesdropping and reverse engineering, Verdult *et al.* [64] were able to prove that vulnerabilities exist in the immobilisers that use Megamos crypto, a proprietary crypto algorithm. This work showed that by not having a pseudo-random number generator in the transponder, the authentication protocol was vulnerable to replay attacks. The internal state of the cipher is only 56 bits, which is smaller than the 96 bit key. From the ciphered state it is possible to compute the predecessor state. There are also implementation issues that enable partial key update and weak key attacks.

2.3.4 Security Requirements of Automotive Systems

We divided the security requirements for automotive systems into two parts. They are the security of the nodes (which are the ECUs) and the security of the communication protocols.

Authentication: Is important in a CAN bus network because of the broadcast nature of the communication. Since any node can transmit and receive messages across the bus, an attacker may conduct his/her attack at any stage of a car's life cycle, to any ECU and affect the operation of the car. By having authentication as a requirement in the network, only valid nodes are able to participate in the communication. Nodes are considered valid if they possess the required cryptographic key. The keys are managed by a valid authority, for example the car manufacturer. Authentication is a requirement for both the nodes and the communications. Entity authentication is a must for the nodes to ensure that they are who they claim to be and are authorised to take part in the communication. Data origin authentication or message authentication verifies the integrity of the data and the source of the data. In a CAN bus, this property is required in communications. It is a security requirement that a CAN bus can verify that the message has not been modified in transit and the receiving parties (all nodes on the CAN bus) can verify from where the data came from.

Integrity: Changing data in an ECU, unauthorised firmware updates and manipulation of messages are all contributing factors to the integrity requirement for the security of automotive systems. To ensure messages are not manipulated, the communication protocol requires integrity. Secure firmware updates or changes and secure data storage are required by the nodes. Therefore the nodes must have integrity as a security requirement.

Freshness: In order to avoid replay attacks, freshness is another security requirement to be considered in CAN bus communication. There is no point in having keys to authenticate the nodes if any malicious attacker can carry out replay attacks by replaying a valid message.

Non-repudiation: Another important security requirement in the CAN bus in order to ensure that no ECUs are reflashed or updated without the consent and acknowledgement of the authorised party. The update will be reflected and no fraud, such as undeclared accidents or undeclared firmware updates, is possible. Once a message is sent by an ECU node, the node cannot deny transmission of the message.

Availability: Any message loss or delay is critical to communication and thus to the operation of the car. The nodes and data associated with them are also required to be available to ensure reliable and complete operation.

Confidentiality: While the content of a node (firmware, key and data) needs to be secured, its confidentiality is a critical requirement to ensure copyright protection (firmware) and secure communication (key). For communication, depending on the application, confidentiality is an option. If authentication is in place, and only an authorised party can access the communication, confidentiality may not be required, unless a party is able to get the messages transmitted across the bus without participating in the communication, for example by accessing a log file of the communications. In this case, communication requires confidentiality to ensure messages are only transmitted between authorised nodes. Confidentiality could also be considered for privacy protection [65].

Authorisation: If a node on a CAN bus is authorised to send a message, then it will be able to identify and authenticate itself during a communication. It will have permission to participate in the communication, for example through a cryptographic key. An unauthorised node will not have permission to send a message, or if it is able to send a message, the message will be discarded/ignored.

Accountability: A node on a CAN bus should be accountable for the messages it sends. The entities involved (for example a workshop) should also be accountable for the communication they participate in. For example, if a workshop performs a service and communicates with the car through its diagnostic tool, the workshop should be responsible for this communication/operation.

The security requirements of automotive systems are as shown in Table 2.3.

Table 2.3: Security requirements of automotive systems

Security requirements	Node	Communication protocol
Authentication	✓	✓
Integrity	✓	✓
Availability	✓	✓
Non-repudiation		✓
Freshness		✓
Confidentiality	✓	
Authorisation	✓	
Accountability	✓	

2.4 Stakeholders in the Automotive Ecosystem

In the automotive ecosystem, there are a number of different entities involved, as shown in Figure 2.2.

The stakeholders are either directly or indirectly involved with the vehicle. They have different functions and interests during the different stages of the life cycle of the vehicle.

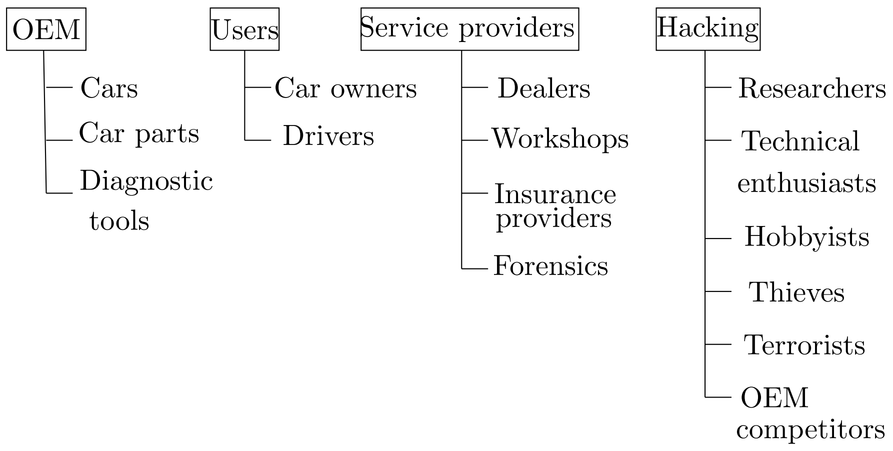

Figure 2.2: Stakeholders in the automotive ecosystem

2.4.1 Types of Stakeholders

Different stakeholders may have different interests and risks [66]. They are divided into two types of accessibility: direct and indirect. The first type, which has a direct relationship with the car or the car parts, includes suppliers, firmware developers, technicians and mechanics at workshops, car agents and dealers, insurance providers, car owners and users/drivers. The second type of entity, which has an indirect connection with the car, are those parties interested in car hacking. They include car manufacturing competitors, hobbyists, researchers, technical enthusiasts, thieves and terrorists.

Attack motivations We summarise four main objectives in performing an attack on an automotive application. The objectives are financial benefit, knowledge gain, satisfaction/recognition and legal benefit. An attacker could want to steal Intellectual Property (IP) or provide/use counterfeit parts. An attacker could also be motivated to compromise the privacy of a user by stealing the user's information. An attacker could want to increase a car's price by conducting data manipulation in the related ECUs. A car owner may want to improve car performance, get cheaper service rates, avoid legal/criminal charges and/or activate after-sale features. Table 2.4 shows the different potential attackers and their objectives.

Table 2.4: Potential attackers and their objectives

Attacker	Objective			
	Financial	Knowledge	Satisfaction/recognition	Legal
Part supplier	✓			
Firmware developer	✓	✓	✓	
Mechanic	✓		✓	
Dealer	✓			
Owner	✓			✓
Insurance agent	✓			✓
Competitor	✓	✓	✓	
Hobbyist/researcher		✓	✓	
Terrorist			✓	
Criminal	✓		✓	✓

Attack methods There are various ways to attack a car, through either logical or physical access. The attacker could perform an attack by impersonation of an authorised entity/tool, manipulation of messages and data, a Denial of Service attack (DoS) or by providing/using counterfeit hardware/software components. Through physical access, an attacker can gain direct access to the in-vehicle networks and/or the ECUs. A logical attack through remote access was seen as something impossible or very unlikely to happen until [12] proved that it is feasible. The most recent and famous attack was conducted through a telematics unit, where the "attackers" were able to control driving operations remotely [15].

Attackers' capabilities Attacks could be performed depending on the capabilities of the attacker. Four main criteria of capability are access level, technical knowledge, technical resources and financial resources. Table 2.5 shows a range of different attackers and their capabilites. The access level could involve physical access or remote access, or the time frame of accessibility. Technical knowledge is the level of expertise an attacker has, while technical resources are the equipment and hardware/software tools that an attacker has. For example, there was an attack by a disgruntled car dealer's employee, who disabled more than 100 cars remotely [67].

Table 2.5: Potential attackers and their capabilities

Attacker	Capabilities			
	Access level	Technical knowledge	Technical resources	Financial resources
Part supplier	Medium	High	High	Medium
Mechanic	Medium	High	High	Medium
Dealer	Medium	High	High	Medium
Owner	High	Low	Low	Low
Insurance agent	High	Low	Low	Low
Competitor	Low	High	High	High
Hobbyist/researcher	High	High	High	High
Terrorist	Medium	High	High	High
Criminal	Medium	High	High	High

2.4.2 Trust Model in Automotive Applications

For automotive applications, there are three entities that are important in the trust model. They are the Original Equipment Manufacturers (OEMs), the application/service providers, and the car owners or drivers. For the security of the components and platforms of the car, which include hardware and software, the OEM is the entity who would want to ensure the security is in place. In terms of the security of services and applications, application and service providers would be interested in ensuring the security and reliability of their systems. On the other hand, car owners and users/drivers might be more concerned about the privacy of the data. Considering the ecosystem of the automotive industry, the car manufacturer holds the most responsibility in terms of the reliability of the car. With this in mind, our proposals consider the car manufacturer as the main entity in ensuring the security of the overall operations of a car including the platform and applications. Explicit trust is given to the car manufacturers. Based on their responsibility and the associated risks, they are given the highest trust level.

Other stakeholders, such as the OEM of the car parts and diagnostic tools, are given an implicit trust, with a medium level of trust. They are usually trusted by the car manufacturer; however, they may have malicious intentions. The users, i.e. the car owners or drivers are given medium to high levels of trust, depending on the application. For example, for a firmware update application, the car owner is given a medium level of trust. He/she may perform the operation for good or malicious objectives. In terms of privacy, the car owner may want to protect his/her privacy for good or even bad reasons. For example, a good reason may be to protect his/her private data such as driving habits; bad objectives may be to avoid being prosecuted/sued for his/her bad driving behaviour.

Service providers such as dealers, workshops, insurance providers, forensic investigators and other related authorities may be given medium and implicit trust. However, privacy is an important issue. A selective set of data, depending on the application, should be given to the respective stakeholders to avoid sensitive data being misused/leaked.

As discussed in [68], the trust model parameters are not only limited to security, but also relate to usability, privacy, reliability and availability, audit and verification mechanisms and user expectations.

2.5 Issues in Connected Cars

Although connected car technology is seen as beneficial, there are some issues that require further consideration.

2.5.1 Data Ownership

There are five main stakeholders for data ownership. They are the car manufacturer, service provider, legal officer, enforcement officer and the car owner. In current automotive industrial practice, car manufacturers have the most access to the data. The service provider, legal or enforcement officer have access to only application-related data. On the other hand, the car owner has limited access to the data. Data access authorisation is commonly related to the purpose of data access. However, this is quite controversial, especially for the car owner.

2.5.2 Privacy

Privacy issues are directly related to the car owner and/or driver. Automotive data such as driving style, location and contact information may disclose more Personally Identifiable Information (PII) about the car owner or driver. Therefore, it is crucial to protect these data.

2.5.3 Reliability of Operations and Data

For automotive systems, reliability of operations is directly related to safety. Therefore, it is crucial to ensure that the operations are functioning appropriately. The data provided by sensors and controllers need to be reliable to ensure safe operations.

2.5.4 Flexibility and Ease of Use

From the point of view of a user, flexibility and ease of use will ensure that the application is easily acceptable.

2.5.5 Security Risks

There are many security risks due to the long life cycle of a car, and the accessibility of the car. The threats and vulnerabilities are discussed in Section 2.3.3.

2.5.6 Control of Service Providers

Giving options for a wider range of service providers opens up a competitive market and could improve the service itself. Car owners would not be tied to specific service providers chosen by the car manufacturer. However, proper security controls must be in place to ensure this flexibility does not introduce vulnerability, especially during the implementation phase. The implementation phase is the critical phase where security could be overlooked by service providers, especially if the guidelines are not specific and ambiguous.

2.6 Mobile Device Integration in Connected Cars

As mobile devices are widely used in our daily lives, they are seen as useful devices in vehicular applications, especially in providing connectivity to the car.

2.6.1 Advantages

Diewald *et al.* described the benefits of mobile device integration in automotive domains [69]. One of the benefits is that users are more familiar with a mobile device compared to a car infotainment system. The lifetime of a car compared to a mobile device and applications is longer, hence it is easier to update applications and/or upgrade a mobile device than to upgrade the infotainment systems. The integration of contacts and related information to any vehicular applications may provide beneficial support, for example for navigation systems. They also suggested additional uses such as preference profile setting/configuration for route and temperature or seat adjustments. Other than that, the mobile device also provides internet connectivity to the vehicle and/or users and passengers.

2.6.2 Network Interfaces

There are two types of connections from the mobile device to the vehicle; wired or wireless.

Wired or wireless: The OBD-II port is a port that interfaces the outside world with the in-vehicle networks [70]. The port can be interfaced with a Wi-Fi, Bluetooth or serial connection using the ELM327 interface [71].

Wired: Mobile devices can be connected to the car through Universal Serial Bus (USB) [72], Mobile High-Definition Link (MHL) [73], High-Definition Multimedia Interface (HDMI) [74], or through analogue audio input/output [69].

Wireless: Mobile devices can be connected to the car wirelessly through WLAN, Bluetooth, NFC, mobile network, or charging ports (only limited support) [69].

2.6.3 Security Issues

Mobile devices are known for their security vulnerabilities [75]. Leinmuller *et al.* evaluated the security of integrating a mobile device with Vehicle-to-X (V2X) communication [76]. They proposed three different configurations of the integration, to manage security at different levels. These functionalities are divided between the mobile device and the on-board unit. They analysed the risk, assets and vulnerabilities of using a mobile device in V2X applications. They concluded that authorisation for the application on a mobile device is a must, regardless where the data is generated. A compromise between flexibility/extensibility and security could be possible, depending on the different levels of security requirements for each different application. Bouard *et al.* considered integrating a mobile device in the automotive networks with physical access to the car [77]. The communication between the mobile device and the ECUs needs to go through a security proxy. They implemented the proposal via two applications running on an Android 3.2 tablet.

There is also a possibility for mobile devices to download malicious applications [78]. A malicious application could disrupt a mobile device by spying, corrupting data and/or interfering with its operations. In this scenario, the mitigation is to ensure that the mobile device is secured in terms of its hardware protection, and that its application store has a certain security verification level to avoid downloading malicious applications. A mobile device should also be protected with an anti-virus application so it can detect unwanted viruses in downloaded applications. The mobile device should also be able to authenticate the source of the applications to be downloaded to ensure that the applications are authentic. Protection should be from hardware to software levels to eliminate this threat.

The use of mobile devices could provide beneficial applications to automotive systems, despite the security issues. Furthermore, the introduction of a trusted execution environment such as ARM TrustZone [79] and Intel SGX [80], could further help to resolve the security issues.

2.6.4 Roles of Mobile Device

A mobile device can be the host to display the infotainment and head unit information, or as the external device that acts as a remote control [69].

2.7 Summary

We discussed the nature of connected cars by introducing the related terms and technologies. Automotive systems were introduced by discussing ECUs and in-vehicle communications. Vehicular applications involve the overall automotive ecosystem; therefore the automotive ecosystem and its stakeholders were discussed in terms of their attack capabilities and motivations. Finally, automotive security issues and requirements are briefly explained.

After providing a basic understanding of the connected car, in the next chapter, we will discuss selected vehicular applications and provide our insights regarding these applications.

Chapter 3

Vehicular Applications

Contents

Three vehicular applications are discussed: firmware updates of ECUs, vehicular forensics and maintenance services logging systems. For each vehicular application, the processes, challenges and requirements are discussed.

3.1 Introduction

There are many vehicular applications for connected cars, consisting of V2V and V2I applications, such as electronic toll collection, autonomous driving and road platoons (groups of vehicles traveling closely together safely at high speed for example by Volvo [81]), hard braking ahead warnings, traffic information and pedestrian detection. A car's life cycle starts at manufacturing, and is followed by selling, use by owner (service/repair, on the road, insurance), reselling and forensics. In this thesis, we focus on three vehicular applications, which are the firmware updates of ECUs, vehicular forensics, and the maintenance services logging system. These three applications are selected to cover different phases of the car life cycle. Firmware updates of ECUs are critical applications in maintaining overall car safety and operation during the whole life cycle of the vehicle. The vehicular forensics and maintenance services logging system are applications that are important to the car owner, and particularly related to the privacy of his/her data. Each application has its own challenges and requirements that need to be addressed.

Chapter organisation The first section, Section 3.2 discusses automotive firmware updates, including the processes, issues, requirements and methods. Next, Section 3.3 describes the automotive forensics in terms of use cases, storage and data retrieval. The next section covers the maintenance services logging system, where types of logging systems are discussed in Section 3.4. Finally, Section 3.5 describes the process, and reasons, for choosing the particular protocol analysis tools for our formal protocol analysis.

3.2 Vehicular Firmware Update

Firmware update is a common process in embedded systems [82, 83, 84, 85, 86]. To implement secure firmware updates in vehicular applications, a specific approach has to be considered, especially as these applications are directly related to safety.

Firmware is the combination of a hardware device and computer instructions and/or computer data that in normal operation can be considered as residing in read-only memory on the hardware device [87]. The software part of modern firmware is stored in the Electrically Erasable Programmable Read-Only Memory (EEPROM) or flash, as compared to old firmware architecture in which it was stored on the Read-Only Memory (ROM). This enables the firmware manufacturer to update its firmware after it is deployed.

Firmware update or ECU reflashing is a potential channel for ECU attack [12, 14, 16, 55]. As mentioned in the previous chapter, ECUs can be attacked either through physical access or via the network. For example, an attacker can replace an authentic ECU with a non-authentic ECU or install malicious or unauthorised firmware to the ECU through the CAN bus. The purpose of firmware updates in automotive applications is to enable car manufacturers to update ECUs for bug fixes, security patches, or to provide performance improvement. Without a well-defined security framework it is possible for a number of threats to be realised (installing malicious firmware or using non-authentic ECUs), potentially with undesirable consequences.

In this section, we present the processes, potential failures, issues and challenges in vehicular firmware update. We then discuss the general requirements and methods of vehicular firmware updates.

3.2.1 Firmware Update Process

Below is a description of the firmware update process as described in [88]. See Fig. 3.2 for the firmware update process for different ECU architectures.

1. The ECU starts with the boot manager, which determines whether to be in flashloader mode or application mode. If a valid application is available, the boot manager will choose the application mode; alternatively it will go into sleep mode. If there is an external request for reprogramming, the flashloader will be initialised (the boot manager will choose the flashloader mode). The flashloader will then load the flash driver once the authentication to unlock the ECU is successful. Refer to Fig. 3.1.

2. In the flashloader mode, if there is an external request for reprogramming, the flashloader will load the flashdriver. The flashdriver contains the routines for erasing and writing to the flash.

   (a) As most current ECUs contain a large flash memory (up to 2048 kB [89]), they are divided into two sections, the data memory and the program memory as in Fig. 3.2(a) and 3.2(d). For these ECUs, the flash driver is loaded into the data memory.

   (b) For small-memory ECUs, there is only one section of flash memory for both program and data memory. For these ECUs, the flash driver is loaded into the Random Access Memory (RAM) as in Fig. 3.2(b), 3.2(c), 3.2(e) and 3.2(f).

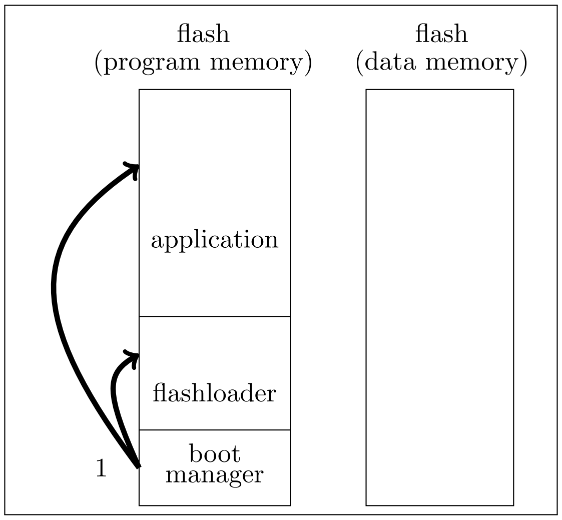

Figure 3.1: Initial state of ECU

(c) The flashdriver can either be loaded from the program memory (the flash-driver is a part of the flashloader) as in Fig. 3.2(d), 3.2(e) and 3.2(f), or loaded from an external device as in Fig. 3.2(a), 3.2(b) and 3.2(c).

(d) If the RAM is small (as low as 128 bytes [89]), the routines are conducted in 2 phases (2a and 2b in Fig. 3.2(c) and 3.2(f).

3. An external programming device, usually known as a diagnostic tool in a workshop, will download and install the new firmware to the flash memory using the flash driver.

3.2.2 Potential Failure

During the firmware update process, there are a number of potential events that might cause the overall process to fail.

Network Disruption

During the download of firmware from the OEM server to the device, any network disruption may cause the firmware update to fail. A retransmission of messages should be in place to ensure complete and successful network communication.

Power Disruption

The device must be properly powered to ensure a successful firmware update. Any power disruption during the update process may cause the process to fail. A failure

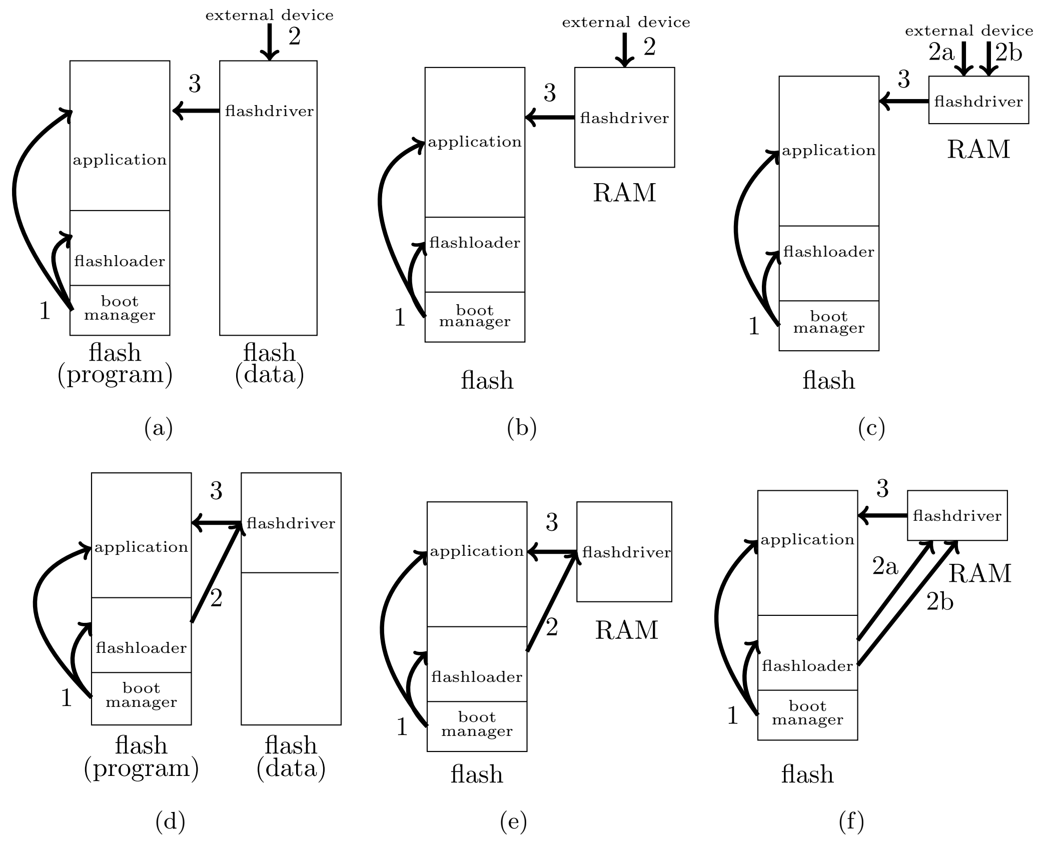

Figure 3.2: Firmware update process
(a) external flash driver with 2 sections of flash memory, (b) external flash driver into RAM, (c) external flash driver into small RAM, (d) internal flash driver with 2 sections of flash memory, (e) internal flash driver into RAM, (f) internal flash driver into small RAM

during the update process may cause the device to be non-functioning and as useless as a brick - hence the term “brick”.

Device Hardware Issues

Before the update process, it is necessary to check the compatibility of the update with the target device; this is usually performed by a program. Among the items to be checked are whether the code downloaded is compatible with the device and whether there is enough memory to store the updated firmware. Ensuring the authenticity of firmware from the OEM can further ensure the compatibility of hardware. OEM may have a number of firmware versions and/or types of firmware but they are customised for specific devices. A corrupted memory on the device could also cause the overall process to fail.

Deliberate Attacks

Any deliberate attacks aimed at causing disruption of the update process, such as network, power or device disruption will cause the update process to fail.

3.2.3 Challenges in Vehicular Firmware Update

Compared to other embedded systems, the vehicular application is quite different in a number of ways. There are multiple ECUs, which may be part of different networks as there are multiple distributed networks in a vehicle. Furthermore, the ECUs are subject to replacement during the life cycle of the vehicle. In addition, the vehicle may be under the control of different entities depending on the stage of its life cycle. The long life cycle of a vehicle must also be considered when proposing secure firmware update solutions for vehicular applications, especially for cryptographic key management and distribution.

Compatibility and Reliability

The compatibility of new firmware with the intended ECU needs to be verified in terms of size, capability and functionality. In addition, all ECUs in the car have to be compatible with each other. Compatibility of the updated ECU with other ECUs in the vehicle is critical to ensure the safety and reliability of the car's operations. If one ECU is not functioning properly, it may affect the operation of other ECUs and thus the operation of the entire car may be affected.

Key Management

Cryptographic key management is commonly handled by the owner of the application. For firmware updates, the cryptographic keys are handled by the OEM of the ECUs. Key management of ECUs needs to consider the life cycle of a vehicle, from manufacturing through to after-sales, which includes repair and upgrade, replacement of parts and resale of the vehicle. The installation of firmware update keys can take place during manufacturing.

Ownership

Ownership can be viewed as the ownership of the vehicle or the ownership of the parts and components of the vehicle, which include the firmware. Although the owner of the vehicle owns the vehicle, the parts and components of the vehicle are under the control of the OEM. Any parts replacements and firmware updates may need the consent of the OEM.

Firmware Recovery

If an update process fails, a firmware recovery procedure needs to be in place to ensure a working ECU, in order to have an operational and reliable vehicle. Updating firmware requires security credentials, i.e. keys. If they are part of the firmware [11], recovery of the keys is crucial. The keys should have proper protection and be able to be retrieved for cryptographic operations. The keys may also reside in another part of non-volatile memory (NVM); for example, a Trusted Platform Module (TPM) or Hardware Security Module (HSM). Among the constraints that limit the firmware recovery process is the memory limitation of the ECU. In addition, the firmware is potentially an intellectual property of the OEM. Thus, it is a requirement by the OEM to be able to protect the confidentiality of the firmware, both that already installed and that about to be installed. There are several options for conducting a firmware recovery process.

1. The old working firmware is stored in a separate memory space within the ECU, and if the process fails, the boot manager will select the last working firmware. However, this option may not be possible due to the memory constraints of the ECU [89, 90].

2. The old working firmware can be extracted and stored in an external device while conducting the update. If the update process fails, the external device will download and reinstall the previous working firmware. However, this option needs adequate security implementation to ensure that the intellectual property of the ECU firmware is protected. There is a possibility for an attacker to perform a rollback attack to rollback to the previous firmware with vulnerability. There should be a mechanism to ensure that re-installation of the old firmware (due to rollback) is not possible during a security update. This is beyond the scope of our work.

Accessibility

Vehicles are potentially easily accessible physically. Although the ECUs are encapsulated in the body of a car, they are still accessible through the network interfaces [12] as mentioned in Section 2.6.2. This makes ECUs vulnerable to attack. For example, a car owner can conduct an update process without authorised access control [16]. An adequate security implementation could limit the accessibility of the firmware update process and thus protect the ECU and the car.

3.2.4 General Requirements for Firmware Update Process

We conclude that there are several general requirements for the overall firmware update process.

Request for Update (GR1)

The ECU receives an external request for reprogramming. A proper mechanism is required to ensure that the request is transmitted from a valid entity (an authorised diagnostic tool). A diagnostic tool is the interface tool used in the workshop in the firmware update process. A malicious diagnostic tool may be used to steal the firmware, to copy the authorisation keys to update the firmware, or to install malicious code. A malicious diagnostic tool may be invalid and non-authentic, or valid and authentic but with malicious intentions. Potential attacks include replay attacks, man-in-the-middle attacks and eavesdropping.

Authentic Flash Driver (GR2)

It is necessary to ensure that the authorised diagnostic tool is the same entity sending consecutive commands. After the authentication of the diagnostic tool, the subsequent commands, which include the loading of the flash driver into RAM or flash data memory, must be guaranteed to originate from the same entity in the same process transaction. If this is not properly checked, a malicious flash driver (code) can be installed to be used in an attack.

Authorised Firmware (GR3)

Authorised firmware is required to ensure compatibility. This can ensure that the updated firmware is able to work and is not malicious.

Authorised Parts (GR4)

Authorised parts, which include the diagnostic tool and ECU, are required to ensure a secure firmware update process and thus the reliability of the car.

Rollback Mechanism (GR5)

If any failure occurs during the update process or the firmware does not work after the update process, a proper rollback mechanism should be in place.

3.2.5 Methods of Vehicular Firmware Update

There are two methods of vehicular firmware update: conventional or over-the-air.

Conventional Firmware Update

A conventional firmware update is performed directly from the OEM to the car through a diagnostic tool as shown in Figure 3.3.

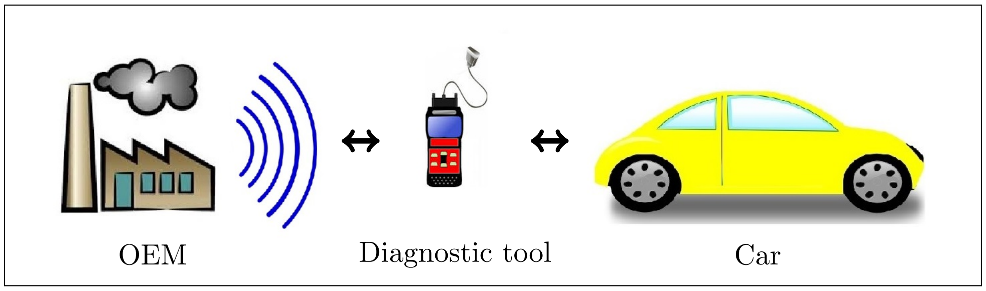

Figure 3.3: OEM trusts diagnostic tool

During the manufacturing of the car, all the ECUs are uploaded with the required cryptographic keys. All the keys are controlled by the car manufacturer, i.e. the keys are managed by an entity entrusted by the car manufacturer. The diagnostic tool at a trusted workshop or dealer is also trusted by the OEM (either the tool is equipped with HSM or the keys are distributed to the tool manually). The tool is then able to communicate with the ECUs in the car and is able to conduct the firmware update process.

Pros: Key distribution is securely managed. Only trusted dealers and workshops are able to conduct the firmware update process.

Cons: Car owners are required to go to a (trusted) workshop to conduct the firmware update process (less flexibility for car owners).

Over-The-Air Firmware Update

An over-the-air (OTA) firmware update is performed with the availability of an internet connection to the car [91]. We divide OTA updates into two methods: direct from OEM (as shown in Figure 3.4) or through a mobile device (as shown in Figure 3.5).

Direct update from OEM to car If ECUs have similar properties to a mobile device, it may be possible to conduct the update OTA, from the server of the OEM to

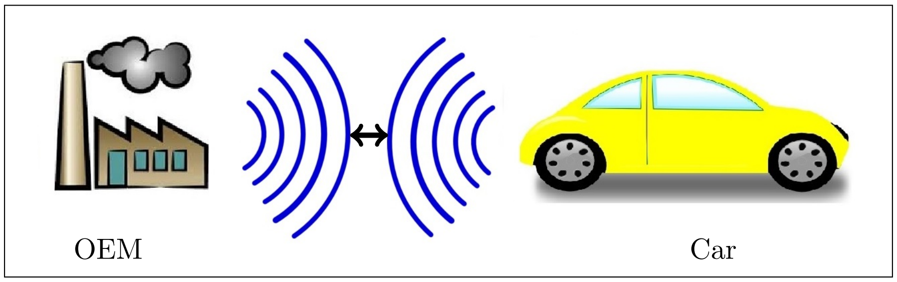

Figure 3.4: Direct update from OEM

the car. The infrastructure required includes the OEM server to generate, store and release the update, and the network to distribute the update. The ECU should also be able to receive the update OTA, i.e. have a network interface and the supporting software to receive and conduct the update.

Pros: Direct connection between OEM and car, without additional interfaces.

Cons: The car is supposed to be online all the time to be able to get notifications and to conduct the update. This increases the risk of being attacked. Furthermore, car owners do not have control of the remote updates performed [92].

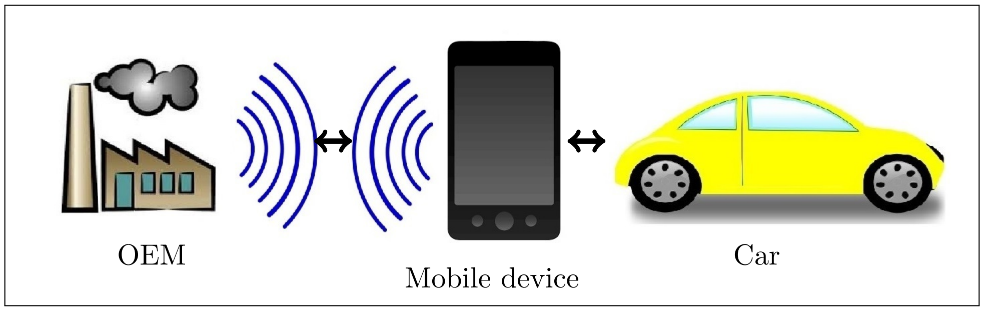

Figure 3.5: Update through mobile device

Update through a mobile device In this concept, the OEM's trust is transferred from the diagnostic tool to the mobile device, which requires a secure mechanism to authenticate the mobile and the user conducting the operation.

Pros: *a*) Distribution of updates is easily accessible, whether to large workshops and dealers, workshops and garages, or to individuals. *b*) The update can be distributed through the mobile operator's network. *c*) In the case of offline updates, the mobile device can first download the update and later apply the update to the car. *d*) There is only a short-term connection between car and outside world, which reduces the attack risk. *e*) An additional medium to conduct rollback.

Cons: A mobile is not a secure communication device, so there is potential for an attack on the mobile device itself.

3.3 Forensics

In this section, another selected automotive application is discussed. Vehicle forensics is becoming an important feature in a vehicle's design and operational life cycle. Interested stakeholders include insurance claim investigators and law enforcement, who are interested in crime and crash incident investigation. In recent years, the forensic feature has been further used by insurance providers and companies providing vehicles to their employees for business-related activities.

In 2002, Duri *et al.* proposed a new framework for automotive telematics that considered security and privacy for end users, service providers and application providers [93]. This work contributed to the introduction of insurance black boxes. They extended the work to investigate the data protection challenges in using telematics in automotive applications [94]. They proposed different levels of privacy policy manager to protect the data, and users can choose which policy level they want for the selected service provider.

The European Commission investigated the use of Event Data Recorders (EDRs) in its projects, Vehicle Event Recording based On Intelligent Crash Assessment (VERONICA), i.e. VERONICA I (2004-2006) and VERONICA II (2007-2009). The VERONICA I project summarised the use of EDRs over the years, along with their requirements and effects [95]. It concluded that EDRs should be further used and available to a wider group of people, not limited to experts. The VERONICA II project evaluated the security issues of EDRs for automotive forensics. In its final report [96], it summarised the security requirements of EDRs, which are confidentiality, integrity, availability and authenticity. EDR stakeholders were divided into three main groups, each with its own interests, which are private, public and joint interests.

Currently, for digital automotive forensics, the two main and most commonly used features are the EDR [97] and the insurance black box that works together with the telematics unit [98]. An EDR is used to store data that is relevant to a crash incident. At least fifteen parameters are stored in order to be recovered during forensic investigation [97], including speed, seat belt status and airbag deployment state. The data is continuously stored and overwritten on the Random Access Memory (RAM) of the

EDR. Data storage to persistent memory (either Electrically Erasable Programmable Read Only Memory (EEPROM) or flash memory) is triggered by a crash-like reading, for example a sudden change in speed. The retrieval of data during the investigation is conducted by reading the content of the EDR, either through the OBD-II port or by physical extraction of the data memory of the EDR. There is however, a possibility that the data fails to be recorded due to electrical failure in the vehicle, which causes insufficient power to write to the EEPROM or flash memory of the EDR. The second possibility for storage failure is that the EDR module is defective. In contrast, an insurance black box continuously transfers the relevant parameters to the insurance company's server through the telematics unit, to monitor driving style. Driving style is used to determine the insurance policy premium rate. A better and safer driving style will result in a lower premium rate. However, there are unresolved issues related to insurance black boxes, which include false data being transmitted to the telematics unit or the server and telematics data not being available.

3.3.1 Automotive Forensics Use Cases

Commonly, forensic data is used during crash incident investigations. However, there are many other use cases where forensic evidence and data logging could be useful. As an example, a current trend adopted by insurance companies is to set the insurance premium rate depending on the driving style. A device called the insurance black box is installed in the car. The device will read parameters such as average vehicle speed, acceleration, braking and cornering behaviours, and time of driving. The driving style of a driver is sent through a telematics unit to the insurance company's server. In another example, available technology such as the Emergency Call (e-Call) service [99] is used during crash accidents, and can call emergency contacts. The service transmits location and time of accident to ensure rapid assistance. For criminal investigations, GPS information could be used [100] to determine the location of a suspect.

Data logging can also be useful for car rentals. For example, Bob goes to the car rental company and is given a car. In the same way that physical body checks are conducted, if Bob can use a diagnostic feature, he can verify the status of the ECUs, and the car as a whole (i.e. digital check) using an application that can interpret the diagnostic data. If the car is in good condition, then Bob agrees to rent it. Otherwise, Bob should notify the company about the vehicle's current condition and he has the choice of whether to rent the car or choose a different car. If a crash or accident happens later, both parties, Bob and the car rental company, will have the evidence to prove the actual situation. Similar to the car rental use case, a potential buyer of a second-hand car can conduct a diagnostic test to ensure that all the units are working correctly as suggested by the seller.

For network intrusion, as an example, an attacker is able to access the in-vehicle network, via the CAN bus network. He injects malicious messages to the CAN bus to cause denial of service (DoS), or to manipulate the operations of a car. The injection of malicious messages into the CAN bus network may cause a change in the normal frequency of messages [11, 54]. By having a data logging system, a driver/owner could be notified about the intrusion. Another example would be a valeting application, where the car key is left with the valet service. Users do not want the valet to compromise their cars, either by trying to steal stored private information, or by trying to trace their future locations by adding a malicious device (for example a USB stick or a malicious ECU) or application to the car.

The final example use case is diagnostic data. If there is any problem with the car, the car owner can conduct a first step diagnostic before visiting a workshop to get the problem solved. This will give the car owner a brief idea of what should be fixed and its estimated cost. Owners can also benefit from assessing their driving style from the stored data, and perhaps modify their driving habits and styles to reduce service costs and likelihood of accidents [101].

The focus of this thesis is the framework to enable the use of this diagnostic application, which could be used to enforce privacy requirements. The application is not the focus of the thesis. With the use of the application in this framework, the user is able to choose which data to share with others.

3.3.2 Forensic Process

The two main processes in forensics are the storage and retrieval of data as discussed below.

Storage of Forensic Data

The ECU could be the storage place for forensic data. However, in the case of total disruption to the car (and/or its units) caused by an accident, the data may not be retrievable. The data in the EDR could be corrupted or changed as a result of bad retrieval techniques [101], or it could be tampered with before storage, e.g. by hacking the CAN bus and injecting malicious data. The cloud or a remote server could be a safe place to store forensic data, as long as the data is recoverable and protected from unauthorised access. The vehicle user should have control of what data is shared with third parties via the cloud or remote server. A mobile device could be another potential place to store forensic data; however, there are concerns about the tamper-resistance of data since a mobile is easily accessible compared to an EDR. A mobile device would be a potential solution for a non-intrusive retrieval method. Compared to data retrieval directly from the car's black box, retrieving data from a mobile device or a cloud would only involve logical digital access rather than physical access. It is a user-friendly method that can be used to provide a first-hand/first impression forensic result, and it reduces the possibility of causing any changes to the actual ECU that could invalidate the evidential data. The mobile could also be used as a backup or complementary unit for the EDR, as there are known problems related to the black box, including failure to record, software and cable faults, OBD port retrieval issues, technical/training problems and permission issues [101]. Compared to an EDR, the mobile, through its Graphical User Interface (GUI) application, can help ensure that the data is readily accessible. The owner or driver can also protect his/her interests by having access to the first-hand forensic data. Forensic data are usually difficult to retrieve (requiring specialised tools and technical expertise if stored in the car). By having the data in the mobile device, the owner or driver has easy access to the data: however, the data must be protected from any malicious tampering.

Data Retrieval of Forensic Data

Parties interested in retrieving forensic data would include law enforcement, lawyers, investigators (for insurance or police), and car manufacturers, as well as the vehicle owner/user. Law enforcement may be interested in the data to determine causes of accidents. Vehicle manufacturers may use this information in order to improve their vehicle designs and performance, and where possible to avoid or minimise the causes of accidents. State and highway officials may be interested in the data to evaluate road conditions and safety. The driver or the car owner might be interested in the vehicle data, but may also attempt to modify or corrupt this data to conceal wrongdoing.

Data stored in a black box can be retrieved in three possible ways: firstly, through physical connection to the OBD-II port, then logically accessing the black box from the CAN bus; secondly, through physical connection to the ECU, then logically accessing the EEPROM or flash memory from the black box operating system and application; and finally, data can also be accessed through physical connection to the ECU's EEPROM or flash memory and conducting a memory dump. If the data is stored on a server, only authorised parties are able to retrieve the data.

3.3.3 Challenges in Automotive Forensics

The challenges in automotive forensics are as follows:

Availability of data The collected data may not be sufficient for forensic analysis, especially for event reconstruction. The limited space for storage [96] contributes to this limited data. It is also necessary to consider technical and security issues. The retrieval of data currently requires expensive specialised tools and expertise [101], although anyone can attempt to access private data via the vehicle network, accessible through the OBD-II port. The availability of data could be compromised if automotive forensics only relies on the availability of the EDR data [97].

Integrity of the data As described in the VERONICA II project [96], there is a window of opportunity for tampering with data. This is between the accident and before the download of data by an authorised party. Integrity and correctness of the stored data cannot be verified in the existing system.

Data ownership There are many stakeholders with an interest in the collected data, including car manufacturers, researchers, insurance companies, legal authorities and car owners. The main challenge is to decide on the ownership of the data. Data may be misused by stakeholders; for example, by the car manufacturer or the insurance company [102]. The access control authorisation for data should really be given to the car owner although certain data is compulsory in order to obtain a service. There is a campaign to try to establish this right [103]. The car owner also needs access to the stored and transferred data to verify that it is correct.

Privacy The existing telematics unit provides connectivity to the car and sends forensic data directly to a server (for example the insurance black box) without the owner knowing what is transmitted; this may violate the privacy of the driver.

3.3.4 Requirements in Automotive Forensics

The requirements for ensuring the provision of a successful automotive forensic system are as follows:

Data availability For forensic analysis, it is crucial for the data to be available. The availability of more data may help to increase the accuracy of the analysis. The retrieved data should be authentic and its integrity should be protected. The data retrieval process must be secured to prevent intentional or unintentional data tampering.

Privacy preserving framework Since automotive forensics may provide identifiable data, a framework that can protect the privacy of the users is required.

3.4 Maintenance Services Logging System

This section discusses another automotive application, which is also related to logging systems. A maintenance logging system allows the car owner to keep the car's maintenance services record updated and hence can reduce the cost of maintenance by avoiding major car breakdowns. In addition, it can add value to the car when it is resold. The potential buyer is assured that the car is in good condition as it is well-maintained. Logging of maintenance services also shows the party who conducted the services, for example, whether they have been conducted by a reliable and trusted workshop or car dealer.

3.4.1 Manual Maintenance Services Logging System

A manual maintenance services logging system is where the process of uploading and storing the records of the services are manual. The workshop will issue receipts to show the list of services performed, or write this information in the car's physical logbook.

In a manual maintenance logging system, a malicious entity can:

(i) Fake a signature to show that the service was conducted by a recognised dealer or workshop. If the process is manual and uses a printed document, the document is stamped as a proof of signature. The stamp can be forged.

(ii) Fake a record to show that a service has been conducted when it has not. Dates can easily be changed or faked.

(iii) Change the list of maintenance services conducted.

With a manual logging system, manipulation can be conducted by the car dealer or the car owner, in order to increase a car's value when reselling it. The owner might request a workshop to falsify the records. The car dealer might falsify the records. An untrustworthy workshop might falsify their list of services, repairs or part replacements to obtain a higher profit.

3.4.2 Automated Services Logging System

In this system, the process of recording and storing the log is automated, mainly operated by the workshop or car dealer. Automated processing can ensure availability of data and ease of use. In the automated logging system, the potential storage locations are electronic data loggers, mobile devices or cloud servers. The electronic data logger resides in the car and is connected to the CAN bus as one of the nodes. The mobile device is an external device (to the car), and it requires a connection to communicate with the in-vehicle network. The log could also be stored on a cloud server.

3.4.3 Challenges in Maintenance Services Logging System

The challenges in maintenance services logging systems are as follows:

Availability of data Since the process of maintaining a logbook is an administrative burden, the availability of the data may be compromised. The car owner normally does not have access to the data, unless it is manually recorded in a log book that he/she keeps.

Integrity of the data The records in a manual logbook can be easily changed if the process is manual. There is a high motivation to change the content of the record, especially when the car is resold.

Data reliability In current implementations, there is no way the user/car owner can verify the correctness of the data about the maintenance/repair being conducted. The car owner cannot validate the services being conducted, and can only trust the information provided by the workshop through the receipts or documents provided.

Data authenticity A potential buyer does not have an assurance that the records in the maintenance log and from the workshops who performed the services are authentic.

3.4.4 Requirements in Maintenance Services Logging System

Storage The process of storing the maintenance data can be improved if it is automated and the data can be verified. Possible storage media could be a dedicated ECU or an external device that is able to retrieve the required data from the car. For example, the data could be stored on a mobile device, and later on a cloud server.

Data security protection The data should be protected in terms of its authenticity and integrity to ensure its correctness and reliability.

3.5 Choosing the formal analysis tools

There are many protocol analysis tools available, such as Tamarin, AVISPA, ProVerif, Athena, CasperFDR and Scyther. Formal protocol analysis is not the main focus of our research. It is for completeness to ensure that our proposed protocols are secure and correct. Based on our brief research on the tools [104, 105, 106], we decided to use CasperFDR and Scyther, mainly because of their ease of use in terms of modeling the protocol and providing scripts and visual output. For example, [106] shows that Scyther is more user-friendly than ProVerif in terms of output (visual output) and usage. In [105], it is shown that ProVerif is the best (compared to Scyther, CasperFDR and AVISPA) in terms of performance, but modeling the protocol is time-consuming.

We chose to use two tools and compare the results obtained. Both tools model security based on the Dolev-Yao adversary model in which the adversary can overhear, intercept and synthesize messages. Although both tools show that the proposed protocols are secure and correct, Scyther provides additional security properties that can be verified: authentication (synchronisation is an additional property in Scyther) and confidentiality.

Furthermore, the output of the attack location and visualisation is shown in Scyther but not in CasperFDR. Scyther is easy to use the process requires the user to create a script and run the verification, then the results of attacks can be shown (and easily understood) in diagrammatic form. Although both tools are popular and publicly available, Scyther is much easier to use as it is easily installed and used on Windows in comparison to CasperFDR, which requires a number of steps to install the GUI and libraries to run on Linux OS.

3.6 Summary

This chapter discussed three of the main vehicular applications. Firmware updates of ECUs could be conducted OTA, either with the need for the car's presence in the workshop, or could be performed at any convenient place with a specified capability. Vehicular forensics and maintenance record systems are related to the car's logging system. For each vehicular application, the processes, challenges and requirements are discussed.

The discussions of the three selected vehicular applications will be useful in the following chapters, where we will discuss the proposed protocols. The discussions provide the rationale for our solutions to the issues mentioned.

Part II

Proposed Improvement on Vehicular Applications

Chapter 4

Firmware Updates

Contents

Firmware updates for vehicular systems are crucial in ensuring the safety and reliability of the car. One of the main industrial projects, the E-safety Vehicle Intrusion Protected Applications (EVITA) project, proposed an over-the-air firmware update protocol. We found some shortcomings and proposed an improved protocol, EVITA+. Our proposed protocol provides additional assurance by considering both security and general requirements to ensure a successful update. These features can help to provide assurance about the reliability and safety of the car.

4.1 Introduction

Secure and updated firmware for ECUs is crucial to the overall security and reliability of the vehicle and its electronic system(s). Therefore, the life cycle of these controllers should be carefully managed. In this chapter, we examine the vehicular firmware updates process and the associated security issues. We analysed the firmware update protocol proposed in the E-safety Vehicle Intrusion Protected Applications (EVITA) project, referred to as the EVITA protocol, which is considered one of the primary industrial efforts in this field and found some potential shortcomings. Based on the analysis, in this chapter we suggest a number of improvements to the EVITA protocol, related with safety and security measures. The proposed improved protocol, also referred to as the EVITA+ protocol, includes a rollback mechanism while preserving the confidentiality of the firmware. The integrity and authenticity of the flash driver are also considered in the EVITA+ protocol. The EVITA+ protocol is implemented to measure its performance and formally analysed using CasperFDR and Scyther, with no known attacks found.

Chapter organisation The first section, Section 4.2 discusses the motivation for studying automotive firmware updates. Next, Section 4.3 describes the related work on automotive firmware updates. The threat model for automotive firmware updates follows in Section 4.4. With this background, we propose a set of security requirements for firmware updates as discussed in Section 4.5. These are followed by our analysis of the EVITA protocol, Section 4.6. Our contribution to improve the EVITA protocol, which is called the EVITA+ protocol, is discussed in Section 4.7. Next, Section 4.8 describes the analysis of the EVITA+ protocol, which includes informal and formal analysis, and our implementation of the protocol.

4.2 Motivations

The E-safety Vehicle Intrusion Protected Applications (EVITA) project proposed a secure firmware update protocol for automotive systems [31]. After analysing the protocol, we found some shortcomings that may cause operational issues. In particular, in a firmware update process, the main issues are the confidentiality, authenticity and integrity of the old and new firmware and the security of the update process. The final stage of the update process should not cause any part of the car to be non-functional 3.2.1 as this could potentially cause an operational and/or safety issue. Therefore, we need a rollback mechanism in case a failure occurs during the firmware update process 3.2.4. We also need to ensure the confidentiality of both the old and new firmware to help safeguard the OEM's (Original Equipment Manufacturer's) intellectual property and to hinder reverse engineering that may aid attackers. Last but not least, we need to ensure the authenticity and integrity of the new firmware.

4.3 Related Work

There have been a number of studies examining firmware updates in vehicular applications.

Nilsson *et al.* identified the need to verify the content of flash memory after the installation process to ensure integrity of the updated firmware [107]. They proposed a protocol to verify the integrity of the new installed firmware. As the new firmware is transferred into RAM, there could be an attack before the installation of the new firmware into the flash. Using the hash chain of each block of firmware, the integrity of the firmware can be verified. The work only considered authentication for external communications, not on-board communications (communications within the vehicle). Confidentiality of firmware in transit over an insecure network and freshness of the communication were not considered in this work.

Adelsbach *et al.* proposed a secure software delivery and installation method using automotive applications as a case study [108]. They identified the model in the software update process and gave specific roles to entities based on their requirements. Their proposal was a secure installation procedure based on public key broadcast encryption and trusted computing. The focus was more on the secure software delivery and distribution with Digital Rights Management (DRM) in the automotive community. The work only considered authentication for external communications, and did not examine in-vehicle communications. Freshness was considered as an assumed requirement during implementation. In another work, they proposed a Trusted Computing Base (TCB) protocol, with three design options [109]. The options were proposed to give different levels of flexibility versus trust level. The three options are: independent tamper-resistant ECU; central tamper-resistant ECU using secure storage; and central tamper-resistant ECU using partially insecure storage.

Patsakis *et al.* investigated security for in-vehicle communication [110]. They proposed extending the use of the immobiliser to authenticate all ECUs. The protocol involved mutual authentication of ECUs with the immobiliser, module voting by ECUs and ticket issuing. The work only considered authentication for on-board communications, not external communications. Secure storage was not considered in this work.

4.4 Threat Model

Table 4.1 shows the different entities and their potential objectives during firmware updates. While the main objectives of firmware updates are performance improvement and bug fixes (including security vulnerabilities), there are also other malicious objectives. These include attacks to prove that there is a vulnerability, to gain knowledge for the hackers own advantage, or to install malicious or non-authentic firmware. These attacks may be conducted for research, monetary benefit, crime, theft or as an act of terrorism.

As shown in 4.1, parts suppliers, firmware developers and car manufacturers perform firmware updates to improve the performance of their devices or to fix bugs in any of the operations. A mechanic from a workshop, or a dealer, performs a firmware update to improve the performance of the car; for example, to improve fuel efficiency. A mechanic from a workshop may perform an update to install non-authentic firmware for monetary benefit, which may be requested by the car owner. An insurance agent may want to perform a malicious update to change the data in the ECU to hide actual data that can be used as forensic evidence. A competitor may want to perform an update to spy on the operations of the car, or to perform reverse engineering, or even to cause a malicious operation that could tarnish the image of its competition. A hobbyist/researcher may perform a firmware update for all three reasons, depending on what area of research/interest he/she is exploring. A criminal might try to play around with updates to steal a car or bypass its security, steal parts for use on other cars and use fake or counterfeit parts for repairs. A terrorist might want to make the vehicle unsafe (so someone crashes), or controllable, or be able to track the vehicle. The owner might want to make the car go faster, or conceal the true mileage and/or origin when selling the car.

As shown in Table 4.2, different entities may be able to conduct the firmware update process depending on their different capabilities. For example, car owners may have little knowledge (expertise) of firmware updates and lack the equipment to conduct the update. Therefore, their capability (combination of expertise and equipment capabilities) is considered as low. The threats to firmware update processes are as follows:

We devised our threat model by evaluating the potential risks related to firmware updates. These risks, as mentioned above, may be related to the motivation for performing the attack, the entities involved in the process and their capabilities.

Table 4.1: Firmware update entities and their objectives

Entities	Objective		
	Improve performance	Fix bugs	Malicious
Part supplier	✓	✓	
Firmware developer	✓	✓	
Car manufacturer	✓	✓	
Mechanic	✓		✓
Dealer	✓		
Owner	✓		✓
Insurance agent			✓
Competitor			✓
Hobbyist/ researcher	✓	✓	✓
Terrorist			✓
Criminal			✓

Table 4.2: Firmware update entities and their capabilities

Entities	Constraints		Capabilities
	Expertise	Equipment	
Part supplier	High	High	High
Firmware developer	High	Medium	High
Car manufacturer	High	High	High
Mechanic	Medium	High	Medium
Dealer	Medium	High	Medium
Owner	Low	Low	Low
Insurance agent	Low	Low	Low
Competitor	High	High	High
Hobbyist/ researcher	High	High	High
Terrorist	High	High	High
Criminal	High	High	High

Obtaining Firmware

Firmware is the main asset in the firmware update process. Without security, the firmware can be easily obtained, either by connecting a Joint Test Action Group (JTAG) [111], using an integrated circuit or circuit board debugging tool to read from the ECU memory, or by reading the content of the new firmware obtained from the OEM. By obtaining the firmware, an attacker is able to learn about the operations of the ECU, and thus the car as a whole. Among interested entities are the car manufacturer's competitors, hobbyists, researchers and criminals.

Reverse Engineering

If an attacker is able to get the firmware image or the corresponding binaries, he might attempt to reverse engineer the operations of the firmware. This is related to the first threat. Therefore the interested entities are the car manufacturer's competitors, hobbyists, researchers and criminals.

Firmware Modification

Following firmware analysis an attacker might attempt to modify its content in order to introduce unauthorised functionality. Motivated entities are the car manufacturer's competitors, hobbyists, researchers, terrorists and criminals.

Obtaining Access Authorisation

Recent versions of vehicles require an authentication of the external device prior to communication with the on-board ECUs. This includes the communication of the diagnostic tool to the vehicle's ECUs. An attacker is motivated to obtain access authorisation in order to conduct further attacks such as a masquerading attack. Motivated entities are car manufacturers' competitors, hobbyists, researchers, terrorists and criminals. Owners and mechanics would also be motivated to obtain access authorisation to be able to modify the car's parameters for performance improvement.

Installing Unauthorised Firmware

A malicious tool or even an authorised tool used with a bad intention may be used to install unauthorised firmware in ECUs. An unauthorised firmware could be malicious firmware to harm the ECU or the vehicle, or even stolen legitimate firmware. A firmware update may be charged for by the car manufacturer, or not, depending on the features of the firmware. A safety/security update may not be charged, while introducing a paid feature in a firmware update may be charged. An attacker could be motivated to steal legitimate firmware to obtain the paid features. Motivated entities are hobbyists, researchers, criminals, terrorists, owners and mechanics.

Unauthorised Device

Using an unauthorised device, in this case a non-authentic ECU in a vehicle, means that if an attacker is able to get the authentic firmware to operate on the non-authentic ECU, this will cause a loss to the ECU manufacturer. It may also cause safety issues.

4.5 Security Requirements

There are many security requirements for firmware update processes, but we discuss the main security requirements based on the threat model and the potential risks related to firmware updates. The security requirements for the firmware update process are as follows:

Mutual authentication (SR1) is required for all the entities involved in the communication. In the firmware update process, the entities involved are the OEM server, the middle interface device (which could be a diagnostic tool or a mobile device) and the ECU. Mutual authentication includes authentication for both external communications and on-board communications.

Secure storage (SR2) is required in the ECU to ensure that the cryptographic keys reside in a secure tamper-resistant device.

Confidentiality (SR3) of the end-to-end communication between the OEM server and the ECU is required to help protect the OEM intellectual property inherent within the firmware and to hinder reverse engineering.

Integrity (SR4) of the communication, especially the firmware, needs to be protected. The end-to-end communications between the OEM server and ECUs need to have integrity protection.

Freshness (SR5) of every new protocol session needs to be in place to ensure that no replay attacks are possible. For example, using a random number or time stamp would provide freshness on every firmware update session. A diagnostic tool should not be able to simply replay messages from previous update sessions.

These are the basic security requirements in the firmware update process. The list is not exhaustive, but they are the main features. Other security requirements include mutual key agreement, trust assurance and privacy.

4.6 OTA Firmware Update by EVITA Project

In this section, we analyse the security solution proposed in the EVITA project [21]. We then highlight the issues of the proposed EVITA protocol.

4.6.1 Protocol Description

The protocol for the EVITA OTA firmware update is shown in Tables 4.4, 4.5 and 4.6, and the notation is defined in Table 4.3. The EVITA protocol uses a dedicated architecture using the EVITA Hardware Security Modules (HSMs). A dedicated Key Master node is configured to store all public and preshared keys of all other ECUs in the vehicle [31]. A formal methodology for this OTA firmware update protocol is described in [112] using AVATAR and ProVerif.

Table 4.3: EVITA protocol notation

OEM	Original Equipment Manufacturer (part manufacturer)
DT	Diagnostic Tool
CCU	Central Communication Unit
ECU	Electronic Control Unit
Mk	Secret nonce
pk_x	Public key of x = DT, CCU or ECU
psk_{ecu}	Preshared symmetric key between CCU and ECU
sk_x	Private/secret key of x = CCU, ECU, DT or OEM
Na	Seed
Smk	$F(Na, sk)$ is a function to compute key in HSM sk is factory preshared symmetric key between ECU and OEM
SSK	Secure Session Key (firmware encryption key)
Fdr	Flash driver
$\{M\}_K$	Message M is encrypted using key K
$Sign_{sk_x}\{M\}$	Sign message M using private/ secret key x
$\mathbb{F}rm_A$	Old firmware
$\mathbb{F}rm_B$	New firmware
MAC_x	AES based Message Authentication Code using key x
ts	Time stamp
Ack	Acknowledgement
fw	Firmware
$\widehat{\sigma_{\mathbb{F}rm_x}}$	Signed $\mathbb{F}rm_x$ by the OEM
=	Equals
$a\|\|b$	a is concatenated with b

Tables 4.4 and 4.5 and 4.6 show the original EVITA protocol. Referring to Table 4.6, ⋆⋆ denotes the assumed protocol (the message was not shown in the original EVITA protocol, but we assumed the existence of the message from the text).

In the protocol [31], the firmware update starts with a remote diagnosis process (Table 4.4). The diagnostic tool (DT) in a workshop communicates with the ECU through the CCU. The remote diagnosis process is to authenticate the DT to the ECU, and to get the ECU information (i.e. version, type, etc.) to the DT. This

Table 4.4: EVITA protocol for remote diagnosis

1. DT	$\rightarrow$	CCU	:	$M1 \| \| Sign_{sk_{dt}}\{M1\}$ $M1 = \{Mk\}_{pk_{ccu}} \| \| ts1$
2. CCU	$\rightarrow$	ECU	:	$M2 \| \| Sign_{sk_{ccu}}\{M2\}$ $M2 = \{Mk\}_{pk_{ecu}} \| \| ts2$
3. ECU	$\rightarrow$	CCU	:	$M3 \| \| Sign_{sk_{ecu}}\{M3\}$ $M3 = Ack \| \| ts3$
4. CCU	$\rightarrow$	DT	:	$M4 \| \| Sign_{sk_{ccu}}\{M4\}$ $M4 = Ack \| \| ts4$

process will output a secret nonce (Mk) to be used for Message Authentication Code (MAC) computation until the completion of the firmware update protocol. The DT will generate the Mk, encrypt it with the CCU's public key and concatenate it with a time stamp. It will pass this message (M1) with DT's signature to the CCU in the form of a signature with appendix. The CCU will verify the DT's signature, decrypt the message Mk with its secret key and store Mk. It will then encrypt Mk with ECU's public key and concatenate it with a time stamp. It will pass this message (M2) with its signature to the ECU. The ECU will verify the CCU's signature, decrypt the message Mk with its secret key and store Mk. The ECU will send a message to the CCU to acknowledge (the completion of storing Mk). The CCU will verify the signature and send a message to the DT to acknowledge completion of the storage of Mk).

Table 4.5: EVITA protocol for ECU reprogramming mode

5. DT	$\rightarrow$	ECU	:	$M5 \| \| MAC_{Mk}\{M5\}$ $M5 = request\_seed \| \| ts5$
6. ECU	$\rightarrow$	DT	:	$M6 \| \| MAC_{Mk}\{M6\}$ $M6 = \{Na\}_{Mk} \| \| ts6$
7. DT	$\rightarrow$	ECU	:	$M7 \| \| MAC_{Mk}\{M7\}$ $M7 = \{Smk\}_{Mk} \| \| ts7$
8. ECU	$\rightarrow$	DT	:	$M8 \| \| MAC_{Mk}\{M8\}$ $M8 = Ack \| \| ts8$

The second stage is to unlock the ECU into reprogramming mode (Table 4.5). The ECU is unlocked into reprogramming mode if the Smk produced by the DT matches the Smk of the ECU. The DT will send a message (M5) to the ECU to request seed, concatenated with the MAC. The ECU will verify the MAC, generate the seed Na and send a message (M6) containing the encrypted seed (using Mk), time stamp and MAC. The ECU will compute the Smk based on the generated seed. The DT will verify the MAC, and generate the Smk based on the received seed Na. It will then send a message (M7) containing the encrypted Smk (using Mk), time stamp and MAC. The ECU will then verify the MAC, and then compare the precomputed Smk with the received Smk. If they match, the ECU is unlocked to enable reprogramming. The ECU will send a message (M8) to acknowledge the DT.

Table 4.6: EVITA protocol for firmware download

9. DT	$\rightarrow$	OEM	:	$M9\|\|Sign_{sk_{dt}}\{M9\}$ $M9 = request\,fw\,encryption\,key\|\|ts9$
10. OEM	$\rightarrow$	DT	:	$M10\|\|Sign_{sk_{oem}}\{M10\}$ $M10 = \{SSK\}_{psk_{ecu}}\|\|ts10$
11. $\star\star$ OEM	$\rightarrow$	DT	:	$M11\|\|Sign_{sk_{oem}}\{M11\}$ $M11 = \{\widehat{\sigma_{\mathbb{F}rm_B}}\}_{SSK}\|\|ts11$
12. DT	$\rightarrow$	ECU	:	$M12\|\|MAC_{Mk}\{M12\}$ $M12 = \{SSK\}_{psk_{ecu}}\|\|ts12$
13. ECU	$\rightarrow$	DT	:	$M13\|\|MAC_{Mk}\{M13\}$ $M13 = Ack\|\|ts13$
14. DT	$\rightarrow$	ECU	:	$M14\|\|MAC_{Mk}\{M14\}$ $M14 = \{\widehat{\sigma_{\mathbb{F}rm_B}}\}_{SSK}\|\|ts14$
15. DT	$\rightarrow$	ECU	:	$M15\|\|MAC_{Mk}\{M15\}$ $M15 = request\,transfer\,exit\|\|ts15$
16. ECU	$\rightarrow$	DT	:	$M16\|\|MAC_{Mk}\{M16\}$ $M16 = Ack\|\|ts16$

The final stage is the download and install process (Table 4.6). The DT will request a (session) firmware encryption key (SSK) from the OEM. The message (M9) is concatenated with its signature. The OEM will verify the DT's signature, generate the session key (SSK), encrypt it with psk_{ecu}, and send message M10 with a time stamp and the OEM's signature. In the proposed EVITA protocol, M11 was not shown clearly in the message. However, we assumed at this point the firmware is being transferred from the OEM to the DT. The DT will verify the OEM's signature and get the encrypted SSK. The DT will send $\{SSK\}_{psk_{ecu}}$ to the ECU without being able to decrypt and get the clear text value of SSK. The ECU will verify the MAC, and having the psk_{ecu}, it will be able to decrypt the SSK. It will then send an acknowledgement to the DT to notify the success of importing the SSK. The DT will verify the MAC, and start sending the encrypted firmware $\{\widehat{\sigma_{\mathbb{F}rm_B}}\}_{SSK}$ concatenated with time stamp and MAC, block by block to the ECU. Since only the corresponding ECU has the key psk_{ecu}, it will be able to decrypt SSK and finally decrypt the firmware to install it to its memory. The ECU will update its ECU Configuration Register (ECR) value with the updated hash chain upon the successful update of each completed block. The DT will send $requesttransferexit$ to denote the last block has been sent. The final hash chain value in the ECR is compared with the expected hash chain value, which may be part of the firmware itself. The ECU will send acknowledgement to the DT upon successful completion of the update process.

4.6.2 Shortcomings

From our informal analysis, there is still a potential attack during the update process. This is by “bricking” the ECU. The motive is to attack the car by disrupting its operation or making it non-functional. “Bricking” of a module is possible by disrupting the content of the ECU or by disrupting the communication between the ECU and the DT while an update process is in progress, as discussed in Section 3.2.2. This issue can be solved by having backup firmware (i.e. the previously working firmware) that can be restored to the ECU if the latest firmware update process fails. However, the confidentiality of the previously working firmware must be protected.

Rollback and Backup Mechanism

The Hersteller Initiative Software (HIS) specification [88] gives a detailed description of a potential flash update process. In its flash driver specification [113], there are 5 routines: initialisation, deinitialisation, erase, write and read. The read routine, however, if called during the update process, is returned with an error code. The HIS specification [88] claims the read routine is not needed for the purposes of performing an update.

Since the RAM of the ECU may be small and not able to store the entire content of the flash driver, the routines can be separated into 2 stages. The first stage contains the initialise and erase routines, and the second stage contains the write and deinitialise routines. For this reason, a backup of the previously working firmware is required. If the update process fails at a certain point in the update, this will cause the ECU to “brick”.

For example, after all the blocks of the ECU flash program memory (containing the application) have been erased, and during the writing of block n, the process fails. Since the update has not been completed, the ECU will be locked to further programming processes. Upon the next reset, if there is no request from the diagnostic tool, the ECU is considered a “brick”. However, the process may repeat and retry the update. The diagnostic tool will likely abort after a few attempts (as a security measure to avoid attempted attacks, or for performance issues, to avoid taking a long time to repeat the process). An update protocol error can occur in normal use; however, it can also be evidence of attack, such as an attempt to load modified malicious firmware. The ECU loader would therefore be expected to have defensive means to block updates after a retry limit has been reached. In this protected state the ECU would need to revert to the last safe operational firmware.

Before the update process takes place, the currently working firmware needs to be stored as a backup in case the update process fails. However, [88] does not support or allow the upload of firmware by reading the ECU content to ensure IP protection.

Flash Driver Security

The flash driver contains the routines for erasing and writing to the flash memory. It is necessary to ensure that no other flash driver can replace the intended flash driver. The flash driver can either be loaded from the flash loader (which is already in the ECU), or from an external device as shown in Fig. 3.2. If the flash driver is a part of the flash loader, platform attestation can ensure that the loaded flash driver is the correct one. However, the flash driver is usually loaded into the RAM of the ECU from an external programming device. This is where an attack can be conducted [16]. Although the final updated content of the firmware is verified, the flash driver content is not. One solution is to have the hash value of the flash driver stored in the ECU flash loader in the ECU Configuration Register (ECR). When the diagnostic tool loads the flash driver, the ECR value is compared. If it matches, the process can proceed; otherwise it will stop and the ECU will be locked. This solution is an alternative for an ECU with a small memory (as low as 2 kB flash [89]). In an ECU with a large memory, the flash driver is already contained in the flash loader. Apart from verifying the integrity of the flash driver, it is also necessary to check that the flash driver is loaded in the same transaction request for update. This will further protect against any other potential attacks such as replay attacks. If the flash driver is used in another transaction, a replay attack can be used as a means to upload malicious firmware.

4.7 Proposed Solution: EVITA+ Protocol

There are a few assumptions and restrictions in the EVITA+ protocol. Firstly, a CCU firmware upgrade is beyond the scope of this work. This is because in this proposal, the CCU is used as the backup storage for the old firmware. For a CCU firmware update, an external device is required to store the backup of the old firmware. Secondly, any ECU replacement should follow the EVITA requirements for key distribution, i.e. there is a central node that stores all the individual ECU's pre-shared symmetric keys and public keys.

In our EVITA+ protocol, we add a number of messages to overcome the shortcomings discussed in the previous section. The EVITA+ protocol is shown in Tables 4.7 and 4.8. Comparing Table 4.6, 4.7 and 4.8, additional messages introduced in EVITA+ are denoted with ⋆.

Table 4.7: EVITA+ protocol for firmware download (part I)

DT	→	OEM	:	$M9\|\|Sign_{sk_{dt}}\{M9\}$ $M9 = request\, fw\, encryption\, key\|\|ts9$
OEM	→	DT	:	$M10\|\|Sign_{sk_{oem}}\{M10\}$ $M10 = \{SSK\}_{psk_{ecu}}\|\|ts10$
⋆⋆ OEM	→	DT	:	$M11\|\|Sign_{sk_{oem}}\{M11\}$ $M11 = \{\widehat{\sigma_{\mathrm{Frm}_B}}\}_{SSK}\|\|ts11$
⋆ DT	→	**ECU**	:	$\boldsymbol{M12\|\|MAC_{Mk}\{M12\}}$ $M12 = Fdr\|\|ts12$
⋆ ECU	→	**DT**	:	$\boldsymbol{M13\|\|MAC_{Mk}\{M13\}}$ $M13 = Ack\|\|ts13$

After the DT requests a firmware encryption key from the OEM (in M9), the OEM replies to the DT with a firmware encryption key (also named the Stakeholder Symmetric Key (SSK)) in M10. The SSK is encrypted with the symmetric key of the ECU shared with the CCU (psk_{ecu}). The DT is not able to get the SSK as the psk_{ecu} is only shared between the OEM and the ECU. The OEM will then transfer its signed new firmware encrypted with SSK (M11), concatenated with its signature. In M12, the flashdriver is transferred from the DT to the intended ECU. It is concatenated with a MAC using a secret nonce (Mk), obtained during the remote diagnosis from the DT. This will ensure that it is in the same transaction request for update from an authorised DT. This will prevent a replay attack as the Mk is random for every session. An unauthorised DT will not be able to introduce a malicious flashdriver as it is not able to participate in the particular communication session. The ECU will verify the MAC from M12 and reply with acknowledgement as in M13, concatenated with a MAC.

Table 4.8: EVITA+ protocol for firmware download (part II)

DT	$\rightarrow$	ECU	:	$M14\|\|MAC_{Mk}\{M14\}$ $M14 = \{SSK\}_{psk_{ecu}}\|\|ts14$
ECU	$\rightarrow$	DT	:	$M15\|\|MAC_{Mk}\{M15\}$ $M15 := Ack\|\|ts15$
$\star$ **ECU**	$\rightarrow$	**CCU**	:	$\boldsymbol{M16\|\|MAC_{Mk}\{M16\}}$ $M16 = \{\widehat{\sigma_{\mathbb{F}rm_A}}\}_{SSK}\|\|ts16$
$\star$ **ECU**	$\rightarrow$	**CCU**	:	$\boldsymbol{M17\|\|MAC_{Mk}\{M17\}}$ $M17 = request\,transfer\,exit\|\|ts17$
$\star$ **DT**	$\rightarrow$	**CCU**	:	$\boldsymbol{M18\|\|MAC_{Mk}\{M18\}}$ $M18 = \{\widehat{\sigma_{\mathbb{F}rm_B}}\}_{SSK}\|\|ts18$
$\star$ **DT**	$\rightarrow$	**CCU**	:	$\boldsymbol{M19\|\|MAC_{Mk}\{M19\}}$ $M19 = request\,transfer\,exit\|\|ts19$
$\star$**CCU**	$\rightarrow$	**ECU**	:	$\boldsymbol{M20\|\|MAC_{Mk}\{M20\}}$ $M20 = \epsilon\{\widehat{\sigma_{\mathbb{F}rm_B}}\}_{SSK}\|\|ts20$
$\star$ **CCU**	$\rightarrow$	**ECU**	:	$\boldsymbol{M21\|\|MAC_{Mk}\{M21\}}$ $M21 = request\,transfer\,exit,\|\|ts21$
ECU	$\rightarrow$	DT	:	$M22\|\|MAC_{Mk}\{M22\}$ $M22 = Ack\|\|ts22$

Referring to Table 4.8, M14 handles the transfer of the SSK to the intended ECU from the DT. Once the ECU receives and decrypts the SSK, it will send an acknowledgement to the CCU (M15).

In the EVITA+ protocol, the objective is to have a backup of the previously working firmware stored in the flash of the CCU. The ECUs come with different memory sizes (RAM, flash). In our proposal, the CCU, having a large memory capacity (up to 2048 kB flash [89]), will store an encrypted version of the last working firmware of the ECU to be updated using SSK. The signed firmware (by ECU) is read out from

Table 4.9: Meeting the general and security requirements of firmware updates

Requirements	[107]	[109]	[110]	EVITA	EVITA+
GR1	○	○	○	●	●
GR2	○	○	○	○	●
GR3	●	●	●	●	●
GR4	○	○	●	●	●
GR5	○	○	○	○	●
SR1	◐	◐	◐	●	●
SR2	○	●	○	●	●
SR3	○	●	●	●	●
SR4	●	●	●	●	●
SR5	○	◐	●	●	●

Note: ○:Not meet, ◐:Half meet, ●:Fully meet

the ECU in encrypted format (M16) to ensure the confidentiality of the firmware and Intellectual Property (IP) protection. The IP contained within the firmware could be very valuable, providing a product differentiator for safety, reliability, performance or economy; and representing a significant investment in research and development by the manufacturer. M16 is also concatenated with a MAC. This is followed by a request transfer exit from ECU (M17) when the final block is transferred. The encrypted new firmware is transferred from DT to the CCU (M18) with a MAC. This is again followed by a request transfer exit to notify the transfer of the last block of firmware (M19). From the CCU, the encrypted firmware is transferred to the intended ECU (M20) also concatenated with a MAC, and followed with a request transfer exit (M21) to notify the ECU of the last block of firmware. The ECU will receive and install the blocks of firmware and send acknowledgement (M22) with a MAC to the DT upon completion of installation.

If at any point in the update, the process fails, the ECU will be restored back to its previous working firmware, thus avoiding any ECU "bricking". Following a successful update process, the flash of the ECU containing the encrypted backup copy of the previously working firmware will be cleared.

The EVITA+ protocol has the advantage of being efficient in terms of memory, storing the temporary back-up firmware in the CCU. This is more efficient compared to each ECU having extra memory storage to store its own back-up firmware. For example, if there were 10 ECUs that needed memory size X to work, an extra 10X memory would be needed to allow for back-ups in each ECU, whereas EVITA+ protocol only needs an extra X of storage in the CCU, as the upgrade is conducted one ECU at a time.

Table 4.9 shows the comparison between related work, EVITA and EVITA+ and how these meet the requirements in Section 3.2.4 in the previous chapter and Section 4.5 in this chapter. Although most of the proposed protocols considered the main security requirements, the general requirements to ensure a successful firmware update process were not considered. The EVITA+ protocol considers all the security requirements, SR1 to SR5, as well as the general requirements GR1 to GR5. The success of a firmware update process depends not only on the security requirements (SR1-SR5) as mentioned in Section 4.5, but also other general requirements (GR1-GR5) as mentioned in Section 3.2.4.

SR1 is mutual authentication. Half meet in SR1 indicates that only one-way authentication (not mutual) is met. SR5 is freshness. In [109], the freshness property is only an assumption.

4.8 Protocol Analysis

In this section, we analyse the proposed protocol in terms of security and performance.

4.8.1 Informal Analysis

The EVITA protocol is designed to prevent attacks as follows:

Obtaining firmware Firmware is the main asset in the firmware update process. By obtaining the firmware, an attacker is able to learn about the operations. Among the interested entities are the car manufacturer's competitors, hobbyists, researchers and criminals. The firmware can be obtained by eavesdropping during the communication between the OEM and DT or during the installation from DT to the car's ECU. In order to prevent this attack, the firmware's confidentiality is protected by encrypting it with the pre-shared key, psk_{fek}. Even if an attacker is able to get the firmware image, any attempt to reverse engineer the operations of the firmware will not be successful. Only the OEM and the ECU, having the psk_{fek}, are able to encrypt and decrypt the firmware.

Firmware modification Following the firmware analysis, an attacker might attempt to modify the firmware content in order to introduce unauthorised functionality. Motivated entities are car manufacturers' competitors, hobbyists, researchers, terrorists and criminals. If the attacker tries to modify the firmware after the download of firmware from OEM to the DT, the ECU will terminate the process once it verifies the incorrect OEM's signature.

Obtaining access authorisation Recent versions of vehicles require authentication of any external device prior to communication with the on-board ECUs. This includes communication of the diagnostic tool with the vehicles' ECUs. An attacker is motivated to obtain access authorisation in order to conduct further attacks such as a masquerading attack. Motivated entities are car manufacturers' competitors, hobbyists, researchers, terrorists and criminals. Owners and mechanics would also be motivated to obtain access authorisation to be able to modify parameters for performance improvement. Since psk_{fek} and psk_{unlock} are pre-shared between OEM and the ECU, they are not easily accessible as they are stored in HSMs. Only encrypted keys are passed from the OEM to the DT.

Installing unauthorised firmware A malicious DT or even an authorised DT with a bad intention may be used to install unauthorised firmware into the ECUs. An unauthorised firmware could be a malicious firmware designed to harm the ECU or the vehicle, or even a stolen legitimate firmware. Motivated entities are hobbyists, researchers, criminals, terrorists, owners and mechanics. Since the DT is authenticated before the start of the installation phase, it is less likely that an unauthorised DT can participate in the protocol and even an authorised DT will not be able to change the firmware as it is signed by the OEM.

Change of OEM server address In an OTA update, a change in the OEM server address (for example Domain Name System (DNS) poisoning) may cause a Man-in-the-Middle (MITM) attack or a Denial-of-Service (DoS) attack. If an attacker is able to change the source address, and the firmware is not protected, the ECU may be updated with malicious firmware or not be able to be updated at all. On the other hand, if the attacker is able to gain access to the OEM server, he would be able to get the firmware and able to conduct reverse engineering or even modify the firmware. Motivated entities are hobbyists, researchers, criminals, terrorists, owners and mechanics. Authentication of the OEM server by the diagnostic tool will prevent this from happening.

Using unauthorised ECU Unauthorised ECUs will not be able to be updated. The OEM is always updated with the latest ECU information from the proposed update process.

In addition, the EVITA+ protocol is designed to prevent attacks as follows:

Denial of service attack Bricking of the ECU by denial of service is prevented by the rollback mechanism proposed in the protocol. If the update process fails, the rollback mechanism will ensure that the previous working firmware is re-installed and running on the ECU. This will ensure that the operations of the car will not be affected by a denial of service attack during the firmware update operation. The integrity of the backup firmware is also protected.

Unauthorised flash driver The flash driver authenticity is verified in the protocol to ensure that only an authorised flash driver is used in the firmware update process.

4.8.2 Formal Analysis

The proposed protocol is formally analysed using CasperFDR and Scyther tools to verify its correctness. The main reason for performing formal analysis using CasperFDR and Scyther in this thesis is for completeness. The EVITA+ protocol is formally analysed using CasperFDR and Scyther tools to ensure the modified protocol does not introduce any security vulnerability.

The CasperFDR tool takes a high-level description of the protocol with its security requirements, and translates the description into the process algebra of Communication Sequential Process (CSP). The CSP description is then verified using Failure Divergence Refinement (FDR) [114]. Similarly, the Scyther tool is an automated tool for the verification of security protocols [115]. It uses an unbounded model checking approach. The protocol behaviour and security claims can be verified from the operational semantics describing the protocol. The Scyther tool provides a graphical user interface that incorporates the Scyther command line tool and a Python scripting interface. Brief introductions to these tools can be found in the appendices (CasperFDR in Appendix A.1, and Scyther in Appendix B.1).

In the EVITA project, the protocol's security objectives are firmware confidentiality, key confidentiality, internal (CCU-ECU) and external (OEM-DT-CCU) authentication. From our CasperFDR and Scyther input scripts, the following security claims were made and verified for the EVITA+ protocol:

1. Confidentiality of the old firmware ($\mathbb{F}rm_A$) during the backup process, the confidentiality of the new firmware ($\mathbb{F}rm_B$) and the confidentiality of the cryptographic keys. The key confidentiality included confidentiality of the secret nonce (Mk, used as the session key for the whole firmware update process), firmware encryption key (SSK), secret key to unlock the ECU (Smk) and all secret keys ($sk_{oem}, sk_{dt}, sk_{ccu}, sk_{ecu}$).

2. Authenticity of all entities involved in the firmware update process (OEM, DT, CCU and ECU as defined in [31]). This includes agreement and aliveness tests as

defined in [116, 117]. In Scyther, an additional authentication property, i.e., synchronisation, is also verified. Synchronisation considers the content and ordering of the messages [117]. Authenticity properties, which include:

- Aliveness between OEM, DT, CCU and ECU.
- Agreement between DT and CCU of Mk, agreement between ECU and CCU of Mk.
- Synchronisation between OEM, DT, CCU and ECU.

In this section, only the improved protocol is discussed using CasperFDR and Scyther scripts. However, we also verified the overall protocol, which includes the remote diagnosis and ECU reprogramming modes. For both tools, we divided the protocol into two different files of input scripts. The scripts are provided in Appendix A.2 and B.2.

Analysis Using CasperFDR

The full script can be found in Appendix A.2.2. The security properties verified are secrecy, aliveness and agreement. The confidentiality property verifies the secrecy of the Mk, $\mathbb{F}rm_A$, $\mathbb{F}rm_B$ and SSK. The aliveness property verifies the aliveness between DT-OEM, DT-CCU and CCU-ECU. The agreement property is to ensure the agreement of Mk shared between DT and CCU and Mk shared between CCU and ECU. The intruder has knowledge of all the entities (OEM, DT, CCU and ECU) and their corresponding public keys.

The script starts with #Free variables declaration, which declares all the variables used in the protocol. It is followed by the #Protocol description. This describes the messages being transmitted (in sequence) during the firmware download, which starts from the request for SSK (i.e. *1.a ->s:request,ts,{request,ts}{SK(a)}*). In *7.b ->c:{m1}{ssk}%frm1,ts, h({{m1}{ssk}%frm1,ts}{mk})*, it shows that agent c is not able to decrypt the message as it does not have the SSK. It will simply pass the message received in the variable frm1. This means the CCU (agent c) is only able to receive the encrypted version of the old firmware and not able to decrypt it. It is the same in *9.s ->a:{m2}{ssk}%frm2,ts,*
{{m2}{ssk}%frm2,ts}{SK1(s)} where DT is not able to decrypt the encrypted new firmware from the OEM.

In the #Processes, all the involved entities in the protocol and their knowledge are declared. For example *INITIATOR(a,s,c,request,mk,requestexit,Fdr) knows PK, SK(a), PK1*, where a is the DT, s is the OEM server, c is the CCU.

The #Specification section declares all the assertions made to verify the security properties. The confidentiality of Mk, $\mathbb{F}rm_A$, $\mathbb{F}rm_B$, secret keys and SSK are declared as Secret(a,mk,[b]), Secret(b,m1,[c]), Secret(s,m2,[b]), Secret(a,SK(a),[a]), Secret(b,SK(b),[b]) and Secret(s,ssk,[b]). As an authentication verification, the aliveness property between DT-OEM, DT-CCU and CCU-ECU, and the Agreement property between DT-CCU and CCU-ECU are verified.

The #Actual variables section describes the names of the actual agents, servers and the actual variables such as FDR (flashdriver), FrmA (old firmware) and FrmB (new firmware). In the #Functions section all the public and secret keys are declared (symbolic PK,SK,PK1,SK1,PK2,SK2). The #System section again declares all the involved entities in the protocol and their knowledge, but with their actual names. For example, *INITIATOR(DT,OEM,CCU,Req,Mk,ReqExit,FDR)*. The #Intruder Information section declares intruder X who has knowledge of all the entities involved and their public keys, and its own public and secret keys, i.e. *IntruderKnowledge=DT,ECU,OEM,X,PK,SK(X),PK1,PK2*.

All the specifications made were verified and no attack was found for any of the assertions.

Analysis Using Scyther

The full script can be found in Appendix B.2.2. The security properties verified were secrecy, non-injective synchronisation, non-injective agreement and aliveness [117]. The secrecy property verifies the confidentiality of the Mk, $\mathbb{F}rm_A$, $\mathbb{F}rm_B$, secret keys and SSK. The non-injective synchronisation property verifies that all parties (OEM, DT, CCU and ECU) know who they are communicating with, and agree on the content of the messages and the order of the messages. Non-injective agreement verifies that all parties (OEM, DT, CCU and ECU) agree on the content of the variables (Mk). The aliveness property verifies that the intended communication partner (DT-OEM, DT-CCU and CCU-ECU) has executed some events.

The script starts with functions declarations. Then, we have macros of messages to make the script neat and easy to follow. Next, the events and claims are made for each role (DT, OEM, CCU and ECU).

For example, for OEM role, the events are *recv_9(DT,OEM,m9)*, *send_10(OEM, DT,m10)* and *send_17(OEM,DT,m17)*, which means OEM receives the macro $m9$ from DT and sends macro $m10$ to DT; later it will send macro $m17$ to DT. Claims are the security properties to be verified. For example, for the OEM role, *claim_x5(OEM,Secret, SSK)* and *claim_x6(OEM,Secret,sk(OEM)* are for confidentiality. Authentication properties are verified through Agreement (such as *claim_x2(OEM,Weakagree)*, *claim_x4*

(OEM,Niagree)), Synchronisation (*claim_x3(OEM,Nisynch)*), and Aliveness (*claim_x1 (OEM,Alive)*).

The default verification setup was used (i.e. five as the maximum number of runs, type matching and finding the best attack with a maximum of ten patterns per claim). The results for all the claims made are verified as "Ok" in the "Status" and "No attacks within bounds" in the "Comments". This means that no attack was found within the bounded statespace, but there could possibly be an attack outside the bounded statespace [118].

We used two tools to compare the features of the tools, and compare the verification results. After using the two tools, we realised that the Scyther tool has an additional property that can be verified, which is the synchronisation property. The synchronisation property is a stronger authentication property compared to agreement, as it is not vulnerable to preplay attacks [119].

4.8.3 Implementation

In this section, we describe the implementation of the updated protocol, called EVITA+. The purpose of implementation was to measure the performance of the protocol, specifically for the constrained devices, i.e., the CCU and ECU. The computation and communication on the server were not part of the implementation, as we were more concerned with the performance of the protocol on the user's side. Furthermore, the performance of the server and communication varies depending on the implementation.

Implementation Platform

This section explains the chosen platform for implementing the protocol.

Rationale of chosen platform Table 4.10 shows the comparison of features between EVITA HSMs and our chosen platform. Although there are actual ECUs available for individual purchase, those supporting the cryptographic algorithms are still very limited, and have very limited technical support. Therefore, we choose a generic microcontroller that supported most of the cryptographic algorithms to implement the proposed protocols. However, the chosen platform does not have an internal HSM, i.e. no internal non-volatile memory and no separate internal processing unit for secure operation and storage.

Our approach to implementation was to observe the computation time on nodes separately from the communication time. This is because communication can occur on

Table 4.10: Features comparison between EVITA HSMs and PIC32MZ

Security features	HSM			PIC32MZ
	Full	Medium	Light	
Algorithms				
ECC	●	○	○	●
RSA	●	○	○	●
AES	●	●	●	●
CMAC/HMAC	●	●	●	●
SHA1	●	●	○	●
SHA256	○	○	○	●
WHIRLPOOL	●	●	○	○
RNG	TRNG	TRNG	PRNG	PRNG
Clock	●	●	●	○
Counter	●	●	○	○
Internal NVM	●	●	◐	○

Note: ○:Not supported, ◐:Optional, ●:Supported

Table 4.11: Application sizes on PIC32MZ microcontroller

	Code size (kB)		
Protocol part	DT	CCU	ECU
Remote diagnostic	447.88	48.6	47.8
ECU unlock	64.8	7.5	65.8
Install	87.2	71.0	66.3

different interfaces, especially for external communication. For internal communication, the DT, CCU, and ECU communicate via the CAN bus.

Nodes The DT, CCU and ECU were simulated using a microcontroller with all the functions required to be an actual ECU with cryptographic engines.

PIC32MZ2048ECM144 [120] was chosen as the implementation platform for DT, CCU and ECU. It is a 32 bit microcontroller with 2048 kB of flash and 512 kB of SRAM, and operates at 200 MHz clock. It supports CAN bus communication, as required in an ECU. The hardware cryptographic engines support the computation of cryptographic algorithms to produce faster performance.

The application sizes on DT, CCU and ECU are as shown in Table 4.11.

Communication The communication time was calculated based on a single message transmission time.

CAN communication All nodes communicating through the CAN bus (DT, CCU, ECU) were simulated using the same microcontroller as mentioned above, i.e. PIC32MZ. In order to observe the communication through the bus, the Microchip CAN bus analyser tool [121] was used. The CAN bus analyser tool decodes the messages through the bus and these messages can be observed via a GUI on the PC.

Programming and Debugging Environment

PIC32MZ The programming language was C using MPLABX IDE from Microchip [122]. It has a compiler, debugger, and programmer. The libraries, for example for CAN and the cryptographic algorithms, are provided by Microchip. For PIC32, all libraries are provided through MPLAB Harmony [123]. The computation performance of the nodes was measured based on the number of cycle counts given by the MPLABX debugger. One cycle count for a 200 MHz clock is equal to 5 ns.

CAN communication The CAN communication was observed through the Microchip CAN bus analyser tool GUI. It provides a GUI to observe the messages going through the CAN, and the user can input data through the CAN. Communication performance was measured using a LeCroy Waverunner oscilloscope [124].

Experiment Setup

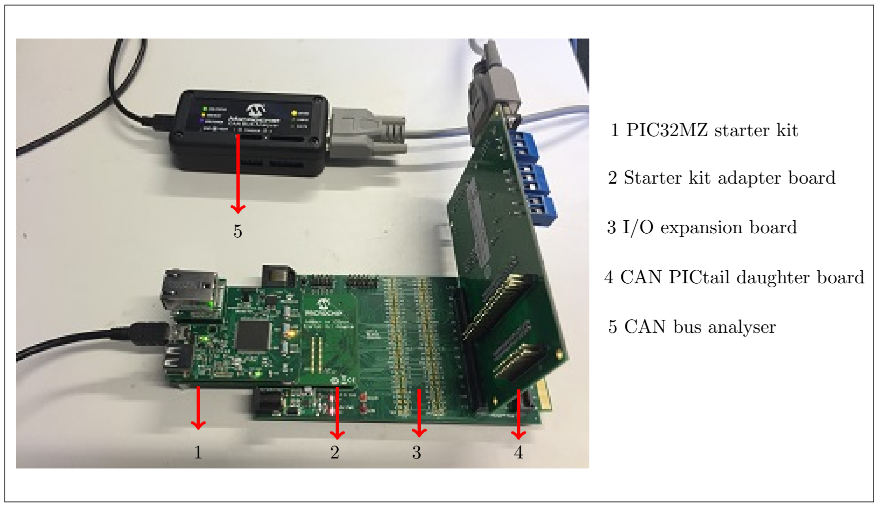

Figure 4.1: Lab setup for nodes communicating through CAN bus

Hardware setup For the CCU setup, the simulation of the messages from and to the DT used the Microchip CAN bus analyser tool [121]. The tool can be used to observe the messages sent from the PIC32MZ microcontroller and also to send messages to it. On the PIC32MZ part, the PIC32MZ2048ECM144 starter kit [125] was connected to a CAN PICtail daughter board [126] through a starter kit adapter board [127] and an I/O expansion board [128]. The CAN PICtail daughter board was then connected to the CAN bus analyser. The setup is shown in Fig. 4.1.

Software setup The size of the flash driver and firmware was set at 160 bytes each for this implementation. This was just an indicative size, whereas a real firmware size could be up to kilobytes or Megabytes. However, performing an update may only mean updating the differential portion that requires the update, not the whole firmware [129]. For MAC computation, HMAC SHA256 was used. RSA1024 was used for digital signatures and public key encryptions. Based on the protocol, the length of a message was more than 8 bytes, hence each message needed to be divided into more than one CAN message due to the limited number of bytes of data per CAN message transmission.

Implementation Results

Table 4.12: EVITA+ protocol performance

Protocol (phase)	Message	Time (ms)					
		Computation			Communication	Total EVITA+	Total EVITA
		DT	CCU	ECU			
Remote diagnostics	1	53.553	53.262		54.081	160.896	160.896
	2		53.313	53.333	3.543	110.189	110.189
	3		13.938	39.672	1.825	55.436	55.436
	4	13.958	39.757		37.387	91.103	91.103
	Total					**417.624**	**417.624**
ECU unlock	5	0.097	0.176	0.078	49.734	50.085	50.085
	6	0.123	0.295	0.172	53.907	54.496	54.496
	7	0.116	0.255	0.139	53.907	54.418	54.418
	8	0.078	0.162	0.084	49.734	50.057	50.057
	Total					**209.057**	**209.057**
Install	9	39.262				39.262	39.262
	10	13.909				13.909	13.909
	11	14.054				14.054	14.054
	12	0.142	0.260	0.117	91.468	91.987	-
	13	0.077	0.157	0.080	49.734	50.048	-
	14	0.102	0.219	0.117	53.907	54.346	54.346
	15	0.078	0.161	0.084	49.734	50.056	50.056
	16		0.161	0.139	2.684	2.984	-
	17		0.077	0.088	0.537	0.703	-
	18	0.138	0.155		45.734	46.027	-
	19	0.089	0.077		24.867	25.033	-
	20		0.138	0.150	2.684	2.973	-
	21		0.088	0.077	0.537	0.702	-
	22	0.077	0.156	0.079	24.867	25.180	25.180
	Total					**417.264**	**196.807**
Total						**1043.944**	**823.488**

The protocol performance is as shown in Table 4.12. The total time for the remote diagnostic phase (M1-M4) was about 417 ms. This is the phase where the DT is authenticated to the car. The second phase, which is the ECU reprogramming mode (ECU is unlocked for reprogramming), took about 209 ms. In the third phase, the total time was about 417 ms. This was the total time excluding the computation time on the OEM server and communication between the server and the DT. This performance will vary according to the size of the flash driver and firmware. Depending on the ECU application, the size of the flash driver and firmware may vary. The communication time can be further improved if CAN FD (Flexible Data-Rate) [130] is used, where one message can contain up to 64 bytes of data, instead of just 8 bytes. CAN FD is a new protocol, and it is not supported in our chosen platform. If CAN FD is used, one message from the protocol; for example, message 1 (in the remote diagnostic phase) can be sent in just 4 messages instead of 33 messages using the standard CAN. Using the same platform, EVITA+ requires about 26% more time compared to the EVITA protocol since EVITA+ has additional messages (messages 12, 13, 16-21).

4.9 Summary

The EVITA project proposed a security protocol for OTA firmware updates. Our proposed protocol provides additional assurance through the rollback mechanism that protects firmware confidentiality. It also ensures secure transfer of the flash driver for ECUs with different memory capabilities. These features can help to provide assurance on the reliability and safety of the car. We present our threat model and outline the general and security requirements to ensure a successful firmware update process.

Chapter 5

FOTA Using Mobile Application

Contents

Security is the main consideration in the EVITA protocol, with limited consideration of the reliability and flexibility of the update process. As we have proposed improvements on the EVITA protocol for reliability of the update process in previous chapters, in this chapter, we investigate the flexibility of the update process. We consider the automotive components under the control of the car manufacturer, especially for the firmware update process, to ensure a reliable, safe and secure car. At the same time, we need to ensure the distribution of firmware updates is flexible and consumer-friendly. This chapter proposes a secure firmware update protocol for automotive systems using a mobile device.

5.1 Introduction

Over-the-Air (OTA) firmware updates are available in some systems, such as mobile networks. Security plays a vital role in ensuring that the firmware update process is successful despite possible threats against it. Therefore mobile devices may be useful to support the OTA firmware update process for other devices such as those used for automotive applications. Using a mobile device as a tool can offer added security features as well as giving flexibility to the process. Automotive security is of high importance as it is critically related to the safety and reliability of the vehicle. We consider an OTA firmware update that eliminates the need for a workshop. We propose a secure Firmware Update OTA (FOTA) protocol to offer flexibility in the firmware update process, while meeting the required security requirements. The proposed protocol is implemented to measure the performance and formally analysed using CasperFDR and Scyther, with no known attacks found.

Chapter organisation The first section, Section 5.2 discusses our motivations for improving automotive firmware updates. Next, Section 5.3 describes the related work on automotive firmware updates. The threat model for automotive firmware update follows in Section 5.4. These discussions support our conclusions about security requirements for firmware updates as discussed in Section 5.5. We then present our second contribution, Section 5.6, which involves automotive firmware updates using a mobile device. Section 5.7 describes the analysis of the FOTA protocol, which includes informal and formal analysis, and our implementation of the protocol.

5.2 Motivations

Firmware updates of automotive subsystems are security-critical, but as yet there is no secure solution that is accessible to all the involved entities. A secure solution with flexibility in the process is important to ensure its usability, especially by car owners.

Car recalls due to software errors that could cause harm/discomfort to passengers [131, 132], may be avoided if a more flexible way of performing the update is available. Providing flexibility to vehicle users to avoid going to car dealers to conduct updates also creates a small cost saving for users as they are not liable for fees for the services provided by car dealers. Aside from the service charge, the trip and the waiting time are also removed.

5.3 Related Work

There are many lessons to be learnt from the firmware update processes in other embedded systems such as smart TVs and set-top-boxes. For an OTA firmware update, a reliable network connection is required between the OEM server and the device. Unavailability of the server may cause the update process to fail [133]. There are a number of ways to conduct firmware updates, both automatically and manually. In a manual process, the user is required to manually check for available updates at a specific website, and store the new firmware in a Universal Serial Bus (USB) to be later installed to the device [134]. Apart from causing inconvenience to users, this adds an unwanted step that might cause the update to fail. For example, as users are not specialists, they might choose not to perform the update, or may not even be aware of the available update for their device. It also leverages attackers to have access to the firmware. However, currently, these devices have direct access to the web server (provided they have an internet connection). This eliminates the manual step of downloading the update onto a USB. The user can then choose whether to conduct the firmware update manually or automatically. In a manual update selection, the user has to manually check for available updates through the menu on the device and select whether to conduct the update or not. In an automatic update selection, the update is automatically pushed to the device [135]. A thorough testing of all versions of devices available in the market needs to be conducted to ensure the update is compatible and will not cause the device to become unresponsive [136].

Firmware Update Over-the-Air (FOTA) for ECUs in the EVITA project [31] uses the diagnostic tool at a workshop. This process requires the car to be in the workshop to initiate the update process.

Flach *et al.* proposed CARMA for personalised tuning [137]. CARMA is a mobile application on Android operating systems that can be used to change some engine parameters (on a single ECU) to improve a car's performance. However, in this preliminary proof-of-concept work, security requirements were not explicitly discussed. Our second contribution extends this in terms of security, and we also consider updates to the whole firmware, rather than only a limited number of parameters on a single ECU.

The efficiency of the update process has been analysed in a few studies [138, 139]. They suggest using delta (only update the part of the software in which changes have been made, not the entire software); compression to minimise the update time; optimised networks; and microcontroller hardware.

In the automotive industry, available FOTA solutions are provided through collaborations between security and update providers. One example is a solution by Novero and Escrypt [140]. There is a server that acts as the software generator and provides key management solutions. The server communicates with the software installer, which is in the gateway ECU. The security in the car is provided by a hardware security module in the gateway.

5.4 Threat Model

The threats are the same as in Section 4.4. An additional threat for this type of firmware update as opposed to conventional firmware updates, where the car does not need to be in an accredited workshop, is:

Change of OEM Server Address

In an OTA update, a change in the OEM server address (for example DNS poisoning) may cause a Man-in-the-Middle (MITM) attack or a Denial-of-Service (DoS) attack. If an attacker is able to change the source address, the ECU can be updated with malicious firmware or may not be able to be updated at all. On the other hand, if the attacker is able to gain access to the OEM server, he can access the firmware and is able to conduct reverse engineering or even modify the firmware. Motivated entities are hobbyists, researchers, criminals, terrorists, owners and mechanics.

5.5 Security Requirements

The security requirements are the same as in Section 4.5. In the OTA process, the diagnostic tool is optional, and therefore its authentication could be ignored if it is not involved.

5.6 Proposed Solution

We decided to choose an architecture with a mobile device as an interface for the FOTA process. In this section, the goals of the protocol are described. Later, the protocol messages are explained in detail. The formal analysis is discussed at the end of this section.

5.6.1 Goals

In this section, we consider the requirements for each entity involved in the FOTA protocol. In a FOTA, there are three entities involved: the OEM, the mobile device and the ECU to be updated. Each entity is responsible for performing certain tasks in the proposed protocol. The tasks ensure that all security requirements are satisfied. For each of the entities, the goal of the protocol is as follows:

1. The OEM is required to
   (a) distribute the updated firmware with confidentiality.
   (b) store updated information related to the car; for example, versions of firmware, parts IDs and associated cryptographic keys.
   (c) ensure secure communication between the OEM and car via the mobile device (including authentication).
2. The mobile device is required to
   (a) pass the encrypted firmware from the OEM server to the car.
   (b) authenticate the parties involved (OEM and car) in the update process.
   (c) pass information from the car to the OEM server and vice versa.
   (d) store the old encrypted firmware from the car as a backup.
3. The ECU is required to
   (a) authenticate the OEM through the mobile device.
   (b) pass an encrypted version of the old working firmware to the mobile device prior to the installation of the update.
   (c) receive the encrypted version of the new firmware from the OEM server (through the mobile device), decrypt it and install it.
   (d) inform the OEM server if any ECU has been replaced, via the mobile device.

5.6.2 Protocol Assumptions

In this section, we discuss the protocol notation (as shown in Table 5.1) and assumptions.

Assumptions

In order to perform the proposed protocol, there are a number of assumptions. We thus limited the scope of our work based on these assumptions.

Table 5.1: FOTA protocol notation

OEM	Original Equipment Manufacturer
CCU	Central Communication Unit
ECU	Electronic Control Unit
MD	Mobile Device
id_x	ID of entity x, x=OEM, mobile device, CCU or ECU
pk_x	Public key of entity x, x=OEM, mobile device, CCU or ECU
sk_x	Private key of entity x, x=OEM, mobile device, CCU or ECU
K_{Nb}	Session MAC key between MD-CCU-ECU
K_{Na}	Session MAC key between OEM-MD
$\{M\}_k$	Message M is encrypted with key k
$sign_{sk}\{m\}$	Message m is signed with private key sk
$MAC_K\{m\}$	MAC of message m using key K
ts	Time stamp
psk_{ecu}	Pre-shared symmetric key of ECU (shared between OEM and ECU)
psk_{fek}	Pre-shared firmware encryption key (shared between OEM and ECU)
psk_{unlock}	Pre-shared symmetric key to unlock the ECU into reprogramming mode (shared between OEM and ECU)
Frm_{old}	Old working firmware
Frm_{new}	New firmware (to be updated)
$=$	Equals
$a\|\|b$	a is concatenated with b

1. Cryptographic keys between the OEM and cars (Central Communication Unit (CCU) and ECUs) are preloaded during the manufacturing stage. The CCU is the central unit that stores all the keys for all ECUs in the same network. All external communications to the car must go through the CCU.

2. HSM-based ECUs are used, i.e. as proposed in the EVITA project.

3. A mobile application will be used to conduct all the mobile device's functions for the OTA firmware update protocol.

4. The mobile application is used for an individual car, i.e. one car per mobile device. For fleet management, or car rental companies, a different registration procedure is required.

5. All ECUs are known to the OEM, i.e. the cryptographic keys are pre-loaded before the ECUs are distributed in the market for parts replacements.

6. Every car has a bar code that can be scanned to obtain the car's details. It may reside on the engine or the dashboard compartment.

7. The ECUs and CCUs are tamper-resistant. They contain HSMs that can provide a layer of security to prevent unauthorised access.

8. The communication channels between the entities are vulnerable to attacks.

9. The mobile device is registered with the car manufacturer. Even if a malicious user uses an authorised mobile device, he would not be able to obtain the firmware.

5.6.3 Protocol Description

The mobile device is seen as a tool to receive notification of available updates and the firmware update itself. When an update is available, the car manufacturer notifies the car owner through the mobile application. The car owner will download the updated firmware into his mobile device. At a later, convenient time, he can download and install the firmware into his car ECU from his mobile device.

Registration

First, we consider the registration process for all the involved parties. In this proposal, we consider the car manufacturer is the trusted party maintaining the registration and the application server for its firmware updates. This could be performed by a trusted third party appointed by the car manufacturer. During the installation and registration, the mobile device will obtain the cryptographic keys for further communications. The required keys are the OEM server public key, the CCU public key and the ECU public key (to verify the signatures of OEM, CCU and ECU respectively). The OEM server and the car will obtain the mobile device's public key. Whenever an ECU is replaced, the mobile device will be updated with the new parameters (this will be discussed in 5.6.3).

1. Mobile device: Firstly, the mobile device needs to install the application, which will be available from the application store. Once the application is installed into the mobile device, the public key of the OEM is obtained.

2. OEM: The unique identification of the car (this could be the vehicle identification number) will be obtained. From the car identification number, car parameters such as the make, model and year of manufacture are obtained. These parameters are important to be able to receive the correct update from the OEM server. This can be done conveniently by scanning the bar code of the car using the mobile device's camera (this is outside the scope of this work). In this phase, the public

key of the CCU and all ECUs' identifications are obtained. The parameters are stored in the mobile device to be later transported to the OEM in order to get the relevant updates.

3. In the event of a change of ownership of the car or mobile device, a deregistration is required. The owner has to notify the registration party.

Notification

This phase is required to ensure the ECU gets the correct update. There are two options for obtaining the update. The mobile user can manually check whether any update is available, or be automatically notified by the OEM.

Table 5.2: Protocol description: OEM-MD Download phase

1.	MD → OEM	:	$M1 \| \| sign_{sk_{md}}\{M1\}$
			$M1 = id_{md} \| \| \{K_{Na}\}_{pk_{oem}} \| \| ts1$
2.	OEM → MD	:	$M2 \| \| sign_{sk_{oem}}\{M2\}$
			$M2 = id_{oem} \| \| ack \| \| ts2$
3.	MD → OEM	:	$M3 \| \| MAC_{K_{Na}}\{M3\}$
			$M3 = id_{md} \| \| RequestDL \| \| id_{ecu} \| \| ts3$
4.	OEM → MD	:	$M4 \| \| MAC_{K_{Na}}\{M4\}$
			$M4 = id_{oem} \| \| \{psk_{unlock}\}_{psk_{ecu}} \| \| ts4$
5.	MD → OEM	:	$M5 \| \| MAC_{K_{Na}}\{M5\}$
			$M5 = id_{md} \| \| ack \| \| ts5\}$
6.	OEM → MD	:	$M6 \| \| MAC_{K_{Na}}\{M6\}$
			$M6 = id_{oem} \| \| \{Frm_{new}\}_{psk_{fek}} \| \| ts6$
7.	MD → OEM	:	$M7 \| \| MAC_{K_{Na}}\{M7\}$
			$M7 = id_{md} \| \| ack \| \| ts7$

Download to Mobile Device

When a new firmware update is available, the update is downloaded into the mobile device in an encrypted version using a firmware encryption key (psk_{fek}). The (encrypted) key to unlock the ECU psk_{unlock} is also transferred to the mobile device. Refer to Table 5.2.

1. The OEM and mobile device will establish a secret session key, K_{Na}. This session key K_{Na} is generated by the mobile device and securely shared with the OEM. It will be used for MAC computations during the whole OEM-MD download phase. The mobile device will sign the message of id_{md}, encrypted K_{Na} and timestamp. K_{Na} is encrypted with the OEM public key, concatenated with a time stamp and the mobile device's signature. The signature is in the format of signature with appendix.

2. The OEM will verify the signature, decrypt the K_{Na} and store it and send an acknowledgement.

3. The mobile device sends a request for the firmware download, RequestDL. It is concatenated with the identification of the respective ECU (id_{ecu}), time stamp and the MAC.

4. The OEM verifies the MAC and sends the key to unlock the ECU into reprogramming mode (psk_{unlock}). The key is encrypted with a pre-shared key between the OEM and ECU (psk_{ecu}).

5. The mobile device verifies the MAC, stores the encrypted $\{psk_{unlock}\}_{psk_{ecu}}$ and sends an acknowledgement.

6. The OEM then sends the firmware (Frm_{new}) encrypted with the pre-shared firmware encryption key (psk_{fek}).

7. The mobile device verifies the MAC, stores the encrypted firmware and sends an acknowledgement.

Download and Install

Refer to Table 5.3 for this phase. Once the new firmware is downloaded in the mobile device, the user can choose to conduct the update at a later convenient time. This is an advantage if the car has no long-range wireless communications of its own.

1. The mobile device and the car will establish a session key K_{Nb}. This session key K_{Nb} is generated by the mobile device and securely shared with the car (CCU and ECU). It will be used for MAC computation during the whole MD-ECU download and install phase. The mobile device will sign the message of id_{md}, encrypted K_{Nb} and timestamp. The K_{Nb} is encrypted with the CCU's public key, concatenated with a time stamp and the mobile device's signature.

Table 5.3: Protocol description: MD-ECU Download and install phase

1.	MD→CCU	:	$M8\|\|sign_{sk_{md}}\{M8\}$ $M8 = id_{md}\|\|\{K_{Nb}\}_{pk_{ccu}}\|\|ts8$
2.	CCU→ECU	:	$M9\|\|sign_{sk_{ccu}}\{M9\}$ $M9 = id_{ccu}\|\|\{K_{Nb}\}_{pk_{ecu}}\|\|ts9$
3.	ECU→CCU	:	$M10\|\|sign_{sk_{ecu}}\{M10\}$ $M10 = id_{ecu}\|\|ack\|\|ts10$
4.	CCU→MD	:	$M11\|\|sign_{sk_{ccu}}\{M11\}$ $M11 = id_{ccu}\|\|ack\|\|ts11$
5.	MD→ECU	:	$M12\|\|MAC_{K_{Nb}}\{M12\}$ $M12 = id_{md}\|\|\{psk_{unlock}\}_{psk_{ecu}}\|\|ts12$
6.	ECU→MD	:	$M13\|\|MAC_{K_{Nb}}\{M13\}$ $M13 = id_{ecu}\|\|ack\|\|ts13$
7.	ECU→MD	:	$M14\|\|MAC_{K_{Nb}}\{M14\}$ $M14 = id_{ecu}\|\|\{Frm_{old}\}_{psk_{fek}}\|\|ts14$
8.	MD→ECU	:	$M15\|\|MAC_{K_{Nb}}\{M15\}$ $M15 = id_{md}\|\|ack\|\|ts15$
9.	MD→ECU	:	$M16\|\|MAC_{K_{Nb}}\{M16\}$ $M16 = id_{md}\|\|\{Frm_{new}\}_{psk_{fek}}\|\|ts16$
10.	ECU→MD	:	$M17\|\|MAC_{K_{Nb}}\{M17\}$ $M17 = id_{ecu}\|\|ack\|\|ts17$

2. The CCU will verify the signature, decrypt the K_{Nb} and store it. It will then encrypt K_{Nb} with the ECU's public key, concatenated with a time stamp and the CCU's signature and send it to the ECU.

3. The ECU will verify the signature, decrypt the K_{Nb} and store it and send an acknowledgement.

4. The CCU will verify the signature, decrypt the K_{Nb} and store it and send an acknowledgement to the mobile device.

5. The mobile device transfers the encrypted unlock ECU key, ($\{psk_{unlock}\}_{psk_{ecu}}$) to the ECU. It is the key that unlocks the ECU to enable it to enter the reprogram-

ming mode.

6. The ECU has its own unlock key, which is only known to itself and the OEM. The ECU will verify the MAC and decrypt $\{psk_{unlock}\}_{psk_{ecu}}$. If it is the correct key, the ECU will switch to reprogramming mode. Otherwise, the process stops here.

7. The ECU transfers the old firmware (Frm_{old}) to the mobile device. This firmware is encrypted using the pre-shared psk_{fek}, concatenated with a time stamp and MAC. The currently working firmware in the ECU (Frm_{old}) is transferred to the mobile device in an encrypted version to ensure its confidentiality. This will be used as a backup for rollback if the installation of the new firmware fails.

8. The mobile device sends an acknowledgement to the ECU once all the blocks of the encrypted Frm_{old} are received. By transferring the old firmware to the mobile device as a backup, it avoids doubling the memory size in all ECUs.

9. The mobile device sends the encrypted firmware update $\{Frm_{new}\}_{psk_{fek}}$ to the ECU.

10. The ECU receives the encrypted firmware update $\{Frm_{new}\}_{psk_{fek}}$ from the mobile device. After the MAC is verified, it decrypts the firmware and installs the updated firmware to the flash of ECU block by block. The firmware is encrypted with a pre-shared firmware encryption key. Only the ECU is able to decrypt the firmware. For every block of installed firmware, the chain of hashes are computed and verified. Any error will terminate the update process and the process will restart again. After three trials, the ECU will roll back to its previous version. If the rollback fails, an error message will be indicated on the mobile application. A replacement of an ECU maybe suggested if there is any issue with memory failure.

ECU Replacement

If any of the ECUs are replaced, the key update protocol needs to be established to ensure further firmware updates on the ECUs are possible.

1. The mobile device (through the CCU) requests the new ECU information (RequestID).

2. The ECU gives its ID to the mobile device (through the CCU).

Table 5.4: Update/ECU replacement phase (between OEM-MD-car)

1.	MD→ECU	:	$M1$ $M1 = id_{md}\|\|RequestID\|\|ts1$
2.	ECU→MD	:	$M2$ $M2 = id_{ecu}\|\|ts2$
3.	MD→OEM	:	$M3\|\|sign_{sk_{md}}\{M3\}$ $M3 = id_{md}\|\|RequestVerifyID\|\|id_{ecu}\|\|ts3\|\|$
4.	OEM→MD	:	$M4\|\|sign_{sk_{oem}}\{id_{oem}\|\|\{M4\}$ $M4 = id_{oem}\|\|\{rnd, psk_{ecu}, pk_{ecu}\}_{pk_{ccu}}\|\|ts4\|\|$
5.	MD→CCU	:	$M5\|\|sign_{sk_{md}}\{id_{md}\|\|\{rnd, psk_{ecu}, pk_{ecu}\}_{pk_{ccu}}\|\|ts5\}$ $M5 = id_{md}\|\|\{rnd, psk_{ecu}, pk_{ecu}\}_{pk_{ccu}}\|\|ts5\|\|$
6.	CCU→ECU	:	$M6$ $M6 = id_{ccu}\|\|rnd\|\|pk_{ccu}\|\|ts6$
7.	ECU→CCU	:	$M7$ $M7 = id_{ecu}\|\|\{\{rnd\}_{psk_{ecu}}\}pk_{ccu}\|\|ts7$
8.	CCU→MD	:	$M8\|\|sign_{sk_{ccu}}\{M8\}$ $M8 = id_{ccu}\|\|ack\|\|ts8$

3. The mobile device requests the OEM to verify the authenticity of the ECU (RequestVerifyID). It is concatenated with the id_{ecu}, time stamp and its signature.

4. The OEM will verify the ECU's ID and will ask the CCU to conduct further verification. It sends a random number (rnd), psk_{ecu} and pk_{ecu}, encrypted with the CCU's public key, to the mobile device.

5. The mobile device passes this message to the CCU.

6. The CCU decrypts the message to obtain the rnd, psk_{ecu} and pk_{ecu}. It then passes rnd and its pk_{ccu} to the ECU.

7. If the ECU is authentic, it can produce the correct $\{rnd\}_{psk_{ecu}}$ and pass the value (encrypted with the received pk_{ccu}) to the CCU. The CCU decrypts and verifies the value sent by the ECU with the precomputed value.

8. The CCU acknowledges authenticity, and sends an acknowledgement to the mobile device.

5.7 Protocol Analysis

In this section, we analyse the proposed protocol in terms of security and performance.

5.7.1 Informal Analysis

The protocol provides the same features as in the previous chapter. It is designed to prevent attacks such as obtaining firmware, firmware modification, obtaining access authorisation, installing unauthorised firmware, changing the OEM server address, using an unauthorised ECU and bricking the ECU (as discussed in Section 4.8.1). In this protocol, the diagnostic tool is replaced by the mobile device.

5.7.2 Formal Analysis

The proposed protocol is formally analysed using CasperFDR and Scyther tools to verify its correctness.

The protocol's security objectives are key confidentiality and internal (CCU-ECU) and external (OEM-MD and MD-CCU) authentication. From our CasperFDR and Scyther input scripts, the following security claims are made and verified:

1. Confidentiality of the secret nonces (K_{Na} and K_{Nb}: used as the MAC keys), symmetric preshared keys (psk_{fek}, psk_{unlock} and psk_{ecu}), the old and new firmware (Frm_{old}, Frm_{new}) and all secret keys (sk_{oem}, sk_{md}, sk_{ccu} and sk_{ecu}).

2. Authentication properties, which include:

   - Aliveness between OEM, MD, CCU and ECU.
   - Agreement between OEM and ECU of psk_{fek}, psk_{unlock} and psk_{ecu}, agreement between MD and ECU of K_{Nb}, agreement between OEM and MD of K_{Na}.
   - Synchronisation between OEM and MD, synchronisation between MD, CCU and ECU.

In this section, only the first part of the protocol is discussed using CasperFDR and Scyther scripts. However, we also verified the overall protocol, which includes the ECU replacement part. The scripts are provided in Appendix A.3 and B.3.

Formal Analysis Using CasperFDR

The full script can be found in Appendix A.3.1. The security properties verified are secrecy, aliveness and agreement. The confidentiality property verifies the secrecy of the K_{Na}, K_{Nb}, psk_{fek}, psk_{unlock}, psk_{ecu}, Frm_{old} and Frm_{new}. The aliveness property verifies the aliveness between MD-ECU and OEM-MD. The agreement property is to ensure the agreement of K_{Nb} shared between MD and ECU, K_{Na} shared between OEM and MD and psk_{fek}, psk_{unlock} and psk_{ecu} shared between OEM and ECU. The intruder has knowledge of all the entities (MD, OEM, CCU and ECU) and their corresponding public keys.

The script starts with the #Free variables declaration, which declares all the variables used in the protocol. It is followed with the #Protocol description. This describes the messages being transmitted (in sequence) during the firmware download, which starts with the MD sending a nonce encrypted with the OEM's public key and a signature is appended (i.e. 1.a → s:a,{kna}{PK1(s)}, ts, {a, {kna} {PK1(s)},ts} {SK(a)}.

In 4.s →a: s, {pskunlock} {pskecu} % ssk, ts, $h(\{s, \{pskunlock\}\{pskecu\}\%ssk, ts\}$ $\{kna\})$, it can be seen that agent a is not able to decrypt the message as it does not have the psk_{ecu}. It will simply pass the message received in the variable ssk. This means that the MD (agent a) is only able to receive the encrypted pskunlock and not able to decrypt it. It is the same in 8.s → a: s, {frmnew} {pskfek} % frm2, ts, $h(\{s, \{frmnew\}\{pskfek\}\%frm2, ts\}$ $\{kna\})$ where MD is not able to decrypt the encrypted new firmware from the OEM.

In the #Processes, all the entities involved in the protocol and their knowledge are declared. For example, INITIATOR(a,s,b,c, kna, requestdl, ack, knb) knows PK, PK1, SK(a), where a is the MD, s is the OEM server, c is the CCU and b is the ECU. The #Specification declares all the assertions made to verify the security properties. The confidentiality of kna, knb, pskunlock, pskecu, pskfek, frmnew and frmold are declared as Secret(a,kna,[a,s]), Secret(a,knb,[a,b]), Secret(s,pskunlock,[b]), Secret(s,pskecu,[b]), Secret(s,pskfek,[b]), Secret(s,frmnew,[b]), and Secret(b,frmold,[b]). As an authentication verification, the aliveness property between MD-ECU and OEM-MD, and the Agreement property between MD-ECU, OEM-MD and OEM-ECU are verified.

The #Actual variables section describes the names of the actual agents, server and the actual variables such as frmold (old firmware) and frmnew (new firmware). In the #Functions section, all the public and secret keys are declared (symbolic PK, SK, PK1, SK1). The #System section again declares all the involved entities in the protocol and their knowledge, but with their actual names; for example, INITIATOR(MD, OEM, ECU, CCU, KNa, RequestDL, ACKs, KNb, RequestExit). The #Intruder Information defines an intruder, X, who has knowledge of all the entities involved and their public keys, and its own public and secret keys, i.e. IntruderKnowledge={MD, OEM, CCU, ECU, X, PK, PK1, SK(X)}.

All the specifications made were verified and no known attack was found for any of the assertions.

Formal Analysis Using Scyther

The full script can be found in Appendix B.3.1. The security properties verified were secrecy, non-injective synchronisation, non-injective agreement and aliveness [117]. The secrecy property verifies the confidentiality of the K_{Na}, K_{Nb}, psk_{ecu}, psk_{fek}, psk_{unlock}, Frm_{old} and Frm_{new}. The non-injective synchronisation property verifies that all parties (MD, OEM, CCU and ECU) know who they are communicating with, and agree on the content of the messages and the order of the messages. Non-injective agreement verifies that all parties (MD, OEM, CCU and ECU) agree on the content of the variables (K_{Na} and K_{Nb}). The aliveness property verifies that the intended communication partner (MD-OEM, MD-CCU and CCU-ECU) has executed some events.

The script starts with function declarations. Then, we have macros of messages to make the script neat and easily followed. Next, the events and claims are made for each role (MD and OEM). The roles are the agents involved in the protocol. For example, for the MD role, the first two events are send_1(md,oem,m1) and recv_2(oem,md,m2), which means the MD starts by sending m1 to OEM and later receives m2 from the OEM.

Claims are the security properties to be verified. The claims are based on the agents' local view of the state of the system. The protocol should ensure that some properties of the global state of the system can be known to the agents based on this local view. For example, for the MD role, claim_a5 (md,Secret,pskFEK), claim_a6 (md, Secret, frmnew), claim_a7 (md, SKR, KNa), claim_a8 (md, Secret, sk(md)) are for confidentiality. This means that the pskFEK, frmnew, KNa, and sk(md) should be secret (not known to the adversary) in every trace of the protocol if the mobile device communicates with any non-compromised OEM [117].

In this script's analysis, the following authentication properties are verified: Agreement (claim_a2 (md,Weakagree), claim_a4 (md, Niagree)), Synchronisation (claim_a3 (md, Nisynch)), and Aliveness (claim_a1 (md,Alive)).

The default verification setup was used (i.e. five as the maximum number of runs, type matching and finding the best attack with a maximum of ten patterns per claim). The results for all the claims made are verified as "Ok" in the "Status" and "No attacks within bounds" in the "Comments". This means that no attack was found within the bounded statespace, but there is the possibility of an attack outside the bounded statespace [118].

5.7.3 Implementation

In this section, we describe the implementation of the proposed protocol. The purpose of implementation is to measure the performance of the protocol, specifically for the constrained devices, i.e. the ECU and the mobile device. The computation and communication on the server are not part of the implementation.

Implementation Platform

This section explains the chosen platform for implementing the protocol. Our approach to implementation is to observe the computation time on nodes separately from the communication time. This is because the communication can occur on different interfaces, especially for external communication. For internal communication, the CCU and ECU communicate via the CAN bus.

Nodes Two platforms are used for the three communicating nodes. There is an additional platform in this protocol compared to Section 4.8.3, which is the mobile device.

CCU and ECU The same implementation platform was used for the CCU and ECU as mentioned in Section 4.8.3.

Mobile device For the mobile device, the application protocol was loaded onto an LG Nexus 5 with a Quad-core 2.3 GHz Krait 400 CPU running on Android 5.1. The mobile device communicates with the CCU through Wi-Fi communication.

The application sizes on CCU and ECU are as shown in Table 5.5.

Table 5.5: Application sizes on PIC32MZ microcontroller

		Code size (kB)	
	Protocol part	CCU	ECU
I	MD-ECU download and install	33.6	85.4
II	ECU replacement	46.9	42.6

Communication The communication time was calculated based on a single message transmission time. Figure 5.1 shows the communication links between mobile device, CCU and ECU.

CAN communication Communication from CCU to ECU and vice versa was through the CAN bus as mentioned in Section 4.8.3.

Wi-Fi communication For the Wi-Fi module on the CCU side, the Wi-Fi G demo board [141] was used. There is a microcontroller (PIC32MX695F512H [142]) on the Wi-Fi module (MRF24WG0MA [143]) that receives the Wi-Fi messages from the mobile device and convert these messages into UART messages, and vice versa.

UART communication Since the Wi-Fi module uses a microcontroller which does not support CAN bus communication, we had an interface module to translate UART messages into CAN messages and vice versa. The PIC18F4580 was used for this purpose. PIC18F4580 [144] is an 8 bit microcontroller with 32 kB of flash and 256 bytes of RAM. It operates with a 16 MHz clock and supports CAN bus and UART communication. In order to observe communication through the UART, the interface module was then connected to an MCP2200 breakout module [145]. The breakout module decoded the messages through the bus and these messages could be observed via RealTerm [146] GUI on the PC.

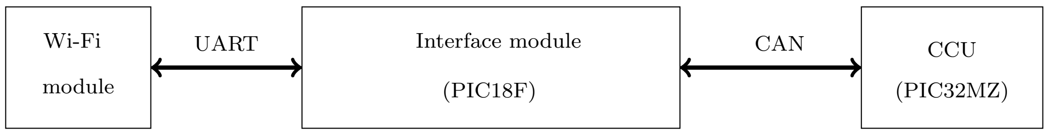

Figure 5.1: CCU's setup for communication

Programming and Debugging Environment

PIC32MZ, PIC18, Wi-Fi module The same programming and debugging environment mentioned in Section 4.8.3 was used for this implementation.

UART communication The UART communication was observed through a RealTerm terminal installed on a Windows 7 machine. RealTerm provides a GUI to observe the messages going through the UART, and the user is able to input data through the UART. Communication performance was measured using a LeCroy Waverunner oscilloscope [124].

CAN communication As mentioned in Section 4.8.3.

Wi-Fi communication The Wi-Fi communication was observed through a web application provided by Microchip library. It provides a GUI to observe the messages going through the Wi-Fi, and the user can input data through the Wi-Fi. The performance of Wi-Fi communication is measured using the "Inspector" feature from the internet browser.

Mobile applications The programming language for the Android applications was Java. The compiler used was an Android Studio 1.5 with JDK 1.8.

Experiment Setup

This section describes the experimental setup, including the hardware and software setups.

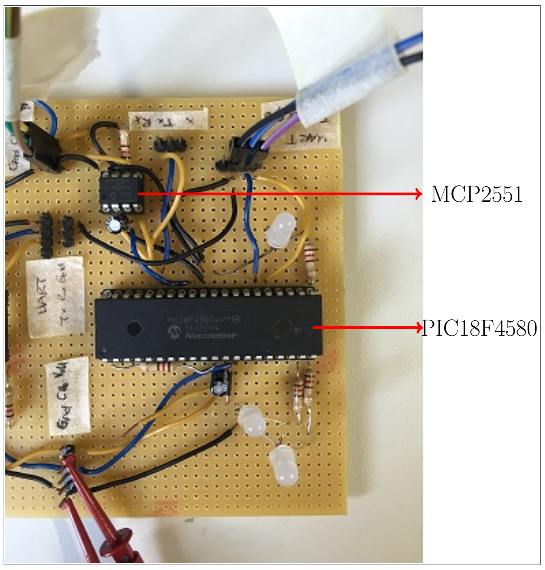

Figure 5.2: Lab setup for interface module CAN-UART communication

Hardware setup The same setup was used for the CCU and ECU using PIC32MZ as mentioned in Section 4.8.3. For the interface module using the PIC18F4580, an additional CAN transceiver, MCP2551 [147], was connected to the PIC18 as shown in Figure 5.2. The interface module was then connected to an MCP2200 breakout module [145] to observe the UART messages.

Software setup The sizes of the flash driver and firmware were set at 160 bytes each for this implementation. For MAC computation, HMAC SHA256 was used. RSA1024 was used for digital signatures and public key encryptions. Based on the proposed protocol, the length of a message was more than 8 bytes, hence each message needed to be divided into more than one CAN message due to the limited number of bytes of data per CAN message transmission.

Table 5.6: FOTA protocol performance

Protocol (phase)	Message	Time (ms)				
		Computation			Communication	Total
		MD	CCU	ECU		
MD-ECU download and install	1	80.015	160.276	92.753	123.117	456.161
	2	3.722	0.433	0.213	29.214	33.582
	3	4.102	0.555	0.265	66.775	71.689
	4	3.891	0.553	0.291	66.775	71.510
Total						**632.951**
ECU replacement	1	2.267	0.005	0.002	8.347	10.622
	2	56.828				56.828
	3	73.632	13.950		25.041	112.623
	4		39.431	14.140	25.041	78.611
	5	22.030	39.282		18.781	80.092
Total						**338.776**

Implementation Results

The protocol performance is shown in Table 5.6. As mentioned earlier in Section 5.7.3, the first part of the protocol (OEM-MD download phase) was not implemented as it involves the communication of the mobile device with the OEM server. There may be different communication channels for this communication, and the server could be on any platform. The performance for constrained devices was the main concern in this implementation. The second part of the protocol, which is the download and install phase between the mobile device, CCU and ECU, took about 633 ms. This performance will vary according to the size of the flash driver and firmware of the ECU application. Finally, the total time for ECU replacement phase was about 339 ms. There was an additional time for the mobile device to communicate with the OEM server and some computational time on the OEM server in this phase. Overall, the communication time can be further improved if CAN FD [130] is used, where one message can contain up to 64 bytes of data, instead of just 8 bytes. However, CAN FD is a new protocol that has been recently introduced, and it was not supported in our chosen platform.

5.8 Summary

In the automotive industry, a secure firmware update process is crucial for safety reasons. This chapter proposes a secure OTA firmware update protocol with the use of a mobile application for the automotive systems. The proposed mobile application will ensure the authentication of all parties involved in the update process and the confidentiality of the firmware. Even if the mobile application is compromised, it cannot reveal unencrypted firmware and associated secret/private keys. The different possible architectures for the OTA firmware update are discussed, and we determined the best architecture based on the security requirements and flexibility of use. The FOTA protocol was analysed using Scyther and CasperFDR and no viable attacks were found.

In conclusion, the security of automotive components requires careful control by the car manufacturer, especially for the firmware update process, to ensure a reliable, safe and secure car. However, the distribution of firmware updates needs to be flexible and consumer-friendly to ensure a high acceptance rate amongst car owners. This chapter proposes a secure firmware update protocol for automotive systems using a mobile device for the convenience of users.

Chapter 6

Vehicle Forensics Using Mobile Application

Contents

Limited features in vehicle forensic systems may lead to unavailability of data or inaccuracy of forensic analysis. Accuracy of forensic analysis can be further improved with a wider range of parameters for the collected data. A secure framework for vehicle forensics is proposed, which still ensures secure data and user privacy. This chapter proposes a secure diagnostic protocol which could be used for vehicle forensics using a mobile device.

6.1 Introduction

Digital forensics is becoming an important feature for many embedded devices [148]. In automotive systems, digital forensics involves multiple Electronic Control Units (ECUs) that are used to support the connected and intelligent vehicle's technology. Digital evidence from these ECUs can be used in forensic investigation and analysis. Such a mechanism can potentially facilitate crash investigation, insurance claims and crime investigation [149]. Issues related to forensics include the authenticity, integrity and privacy of the data [150]. In this chapter, the security of the forensic process and data in automotive systems is analysed. We propose an efficient, secure, privacy-preserving and reliable mechanism to provide a forensics data collection and storage process. A diagnostic application for smart phones, DiaLOG, is incorporated in the proposed process, which uses a secure protocol to communicate the collected forensic data to secure cloud storage. The proposed protocol for communicating forensic data is implemented to measure performance results and formally analysed using CasperFDR and Scyther, with no known attacks found.

Chapter organisation The first section, Section 6.2 discusses the motivations for our attempt to improve automotive forensics. Next, Section 6.3 describes related work on automotive forensics. The threat model for automotive forensics follows in Section 6.4. From this basis, we present the security requirements for automotive forensics as discussed in Section 6.5. Our contribution to improving the existing implementation of automotive forensics, which includes an Android application called DiaLOG and a secure protocol, is discussed in Section 6.6. Next, Section 6.7 describes the analysis of the proposed protocol, which includes informal and formal analysis, and our implementation of the protocol.

6.2 Motivations

Digital forensics requires reliable and tamper-protected data storage. Since more data improves accuracy in any investigation, a secure and reliable forensics solution is required, which is able to store all the necessary data to support a forensic investigation without sacrificing user privacy. The authorised user should have control over what data is to be shared with a particular service provider or other third parties. Easy retrieval of forensic data is also desirable, without sacrificing its integrity and accuracy.

In this chapter, we consider solutions to the following problems:

(i) The current use of Event Data Recorders (EDRs) and insurance black boxes in forensic evidence provides limited features [101, 151]. The EDR gives a restricted number of parameters for analysis, whereas the insurance black box and telematics units do not protect user privacy.

(ii) Although certain data is compulsory to obtain a service, users do not have control of the transmitted data and cannot access it.

(iii) Users are therefore unable to verify the correctness of the data being transmitted. Most forensic data need to be interpreted, using an application, for a non-technical user to understand them.

There are a number of issues related to automotive forensics. One issue is the privacy of data. The existing telematics unit provides connectivity to the car and sends forensic data directly to a server (for example an insurance black box) without the car owner knowing what is transmitted. The access control authorisation of data should really be given to the car owner although certain data is compulsory to obtain a service. The car owner would therefore have a choice in selecting the service provider. There is a campaign to try to establish data ownership rights for car owners [103]. The car owner also needs access to the data (stored and transferred) in order to verify that the data being transferred is correct. For this purpose, the data should be interpreted for the user to understand them, which could be achieved by means of an application. It is also necessary to consider technical and security issues. The retrieval of data currently requires expensive specialised tools and expertise [101], although anyone can attempt to access private data via the vehicle network, accessible through the OBD-II port. Additionally, the integrity and correctness of the stored data cannot be verified in the existing systems. Finally, the availability of data could be compromised if automotive forensics relies solely on the availability of EDR data [97].

6.3 Related Work

Nilsson *et al.* discussed performing forensics on in-vehicle networks [152]. They discussed an attacker model and requirements for detection, collection and event reconstruction. According to them, features such as diagnostics, firmware update, Vehicle-to-Vehicle (V2V) and Vehicle-to-Infrastructure (V2I) communications are desirable and could be achieved wirelessly, but these features introduced the potential for cyber-attacks. In their forensic proposal design, their goals are: to detect events in the vehicle (which require a device to detect, notify and store the forensic evidence); to answer the required forensic questions (from the collected forensic data); and to obtain the current state of the firmware version. In order to do this, a list of hashes of current firmware installed on all ECUs needs to be accessible. Their proposal described what to do, without providing any practical implementation. During the presentation phase in forensics, conclusions can be made from both the physical evidence from EDR data and digital evidence through the network.

Hoppe *et al.* investigated route reconstruction forensics in a hit-and-run scenario [153]. They proposed two methods, i.e. manual and semi-automated. Firstly, a Global Positioning System (GPS) receiver is installed in the car and connected to the ECU through the CAN bus. Any communication is logged in the data logger. The directions for navigation are displayed on the instrument panel cluster for safety and comfort. They propose using this feature to provide forensic evidence by enabling route reconstruction. In the manual method, the data from the data logger is manually analysed, and optimisation is conducted by filtering data that are potentially relevant to the incident. The semi-automated method involves connecting a probe to the navigation unit and to a Graphical User Interface (GUI) to show the data read from the navigation unit. It is an invasive method since a physical connection to the unit is required to get the relevant data. In order to log all communications to the data logger, a large memory space is required. For example, according to [154], there are 63 CAN IDs just to identify the different operations of the car. A single operation is represented by a single CAN ID. As an example, just for the sideways acceleration sensor, there are 35 frames per second. This would require a large memory size for logging (about 20 kB per hour just for one operation). Furthermore, for a car without a navigation system or a data logger, there may be costs for installation of additional equipment.

Kowalick discussed in detail the unaddressed issues regarding automotive EDRs [150] as seen by the National Highway Traffic Safety Administration (NHTSA), which is part of the US Department of Transportation. Among the issues are EDR data ownership; authenticity; security at the time of the crash and the chain of custody after the crash incident; tampering and manipulation; how data can be used for civil/criminal proceedings; police access authorisation for the data; the possibility of developing EDR into a driver-monitoring tool; and third party access authorisation.

In 2010, Al-Kuwari *et al.* investigated the feasibility of performing live forensics using electronic components in the vehicle [149]. They considered the possibility of collecting relevant forensic data via multiple ECUs with different functions, different networks and buses, sensors and multiple applications (such as adaptive cruise control, lane-keeping assist, parking assist, blind spot monitoring, head-up display and night vision, telematics and multimedia, navigation, and occupant sensors). They proposed that forensic data could be collected using devices called collectors. These collectors are installed or attached to ECUs or other components to capture communication through the networks. In their proposal, they make the assumption that privacy is ensured by legal enforcement officers following proper procedures.

6.4 Threat Model

As mentioned in the related work section (Section 6.3), we develop our threat model to mitigate potential risks such as tampering with data and attacks on access privilege and privacy. In automotive forensics, assets to protect include read and write access authorisation and the authentication, integrity and privacy of data. Potential attackers are untrustworthy workshops, car owners/users, investigators and hackers with financial motivation.

A malicious entity:

(i) can access the CAN bus and manipulate the content before storage.

(ii) can access and manipulate the content after storage.

(iii) cannot break well-established cryptographic algorithms.

(iv) can corrupt data in transit and in storage.

A number of possible attacks on automotive forensics systems can be conducted, as follows:

Denial of Service (DoS) attack: To cause availability issues, where data stored is not able to be retrieved, or data is not able to be stored. Denying access to data to an authorised party is also a method of DoS.

Impersonation attack: To impersonate an authorised party to conduct further attacks; for example, an attacker impersonating an authorised person to access data during an investigation to manipulate the content, or a device impersonating an authorised tool to access data during storage (i.e. CAN bus manipulation).

Data manipulation attack: To change the content of the forensic data, by changing the data before or after storage, or during the retrieval process. Attacks can be performed mechanically by simply destroying the black box and its data contents. Attacks might also be mounted electronically, which can cause disruption or change to data. Attacks through malicious CAN bus messages could distract the driver, or hack the engine or braking operation, or cause a crash and then erase all traces from the black box.

6.5 Security Requirements

Although there are many security requirements for forensics, we discuss the main security requirements based on the threat model and the potential risks related to vehicle forensics as discussed in Section 6.4. We determined these requirements based on the focused work on the current limitations/vulnerabilities of EDR as discussed in Section 6.3.

Integrity: It is crucial to ensure that the data stored is not being tampered with or corrupted.

Authenticity: Data must be authentic and the person handling the data (for example a forensic investigator) is authorised.

Availability: Ensuring all the required data is available for investigation, and updated regularly.

Reliability: Having a backup device for forensic data storage can increase the reliability of the forensic system.

Privacy: It is important to protect the privacy of the car owner, especially when handling privacy-related data such as driving habits.

6.6 Proposed Solution

Commonly, during vehicle forensic investigation, the EDR, the infotainment unit and other ECUs are analysed [97]. In this proposal, the mobile application, DiaLOG, is the backup data storage for the EDR and other related data that might be of interest for forensics.

Our proposal considers using recent developments in car applications, which require a security module for each ECU to conduct cryptographic operations [21] and thus provide platform security. This suggests that any node communicating through the CAN bus is required to have access authorisation in order to send or receive messages. In our proposal, the mobile device acts as a communicating node through the CAN bus, and so requires access authorisation.

To conduct a diagnostic on the car, the mobile device is connected to the OBD-II port via Wi-Fi or Bluetooth. Once connected, the mobile device is authenticated, to determine whether it is authorised to retrieve the requested data. Once authenticated,

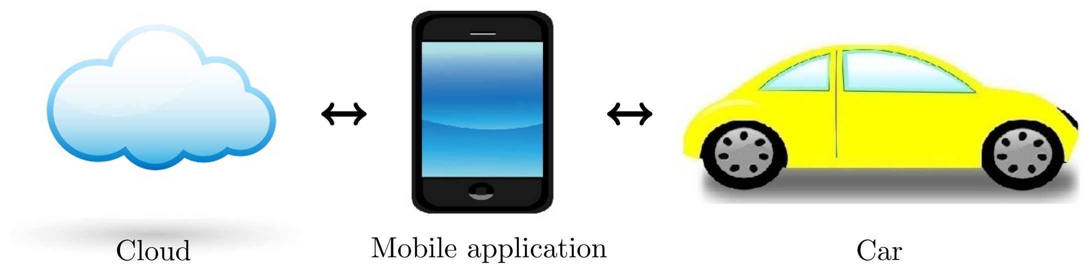

Figure 6.1: Architecture of the proposed forensic framework

the mobile device is connected to the CAN bus, and is able to access the required data. The main idea of the DiaLOG application is to read the DTCs (Diagnostic Transmission Codes) and log them securely. The DTCs can be read by the user of the DiaLOG application, and from there, the user is aware of the car's condition and state.

6.6.1 Assumptions

The proposed protocol relies a number of assumptions. We were thus able to limit the scope of our work based on these assumptions. The mobile application is installed on a mobile device. For an authorised car owner, the mobile device is available for investigation. The data is always automatically transmitted to the phone whenever connection is established, and later to the cloud. If data is not updated after a certain time, the owner will be notified. Finally, the cloud is securely managed. A user is authenticated to access the cloud server, and only authorised users have access to the data. However, even if an attacker is able to get access to the data in the cloud, the integrity and confidentiality of the data could not be compromised if using our protocol.

6.6.2 The DiaLOG Application

The architecture of the proposed framework with mobile application and cloud-based backup storage is shown in Fig. 6.1. A mobile device with the DiaLOG application can log the latest vehicle operations. Once connection between the mobile device and the car's CCU is authenticated, the mobile device will request the data from the CCU. The CCU will get the relevant data from the expected ECU, and reply to the mobile device. The data on the mobile device is then uploaded to the cloud when a suitable network connection is available. From this framework, the forensic investigators have the option to get the data from three different sources: the car EDR, the mobile device or the cloud. The car owner (having the DiaLOG application and access to the forensic data) has control of what data to share with third parties. Certain data is compulsory to obtain a service and the car owner will need to give access to the service provider. However, he/she has access to the transmitted data and can verify the correctness of data. The proposed architecture safeguards the privacy of the car owner/driver, in a way that is not possible with the current system that transmits data via the telematics unit. The keys for the mobile device are stored in a secure memory; for example on a secure element, or the mobile device could be supported by TrustZone. The keys for the CCU are stored in the HSM of the CCU.

Registration

First, we consider the registration process for all the involved parties. In this proposal, we consider the car manufacturer is the trusted party maintaining the registration of the car owner's mobile device. This could be performed by a trusted third party appointed by the car manufacturer. During installation and registration, the mobile device of the car owner will obtain the cryptographic keys for further communications. The mobile device and the CCU will share a set of keys/data. The required keys/data are the symmetric key shared between the mobile device and the CCU, the identification number of the CCU and the identification number of the mobile device. In the event of change of ownership of the car, or the mobile device, the car owner is responsible for performing the deregistration.

Authentication Phase

In order to use DiaLOG with a mobile device, the mobile device must be authorised. Only an authenticated mobile device is given permission to access the data from the car, and most importantly, to connect with the car's internal network. Authorised devices are divided into two different levels: basic or full authorisation. These levels will be further explained in Section 6.6.3.

Diagnostic Phase

In this phase, the mobile device is connected to the vehicle through a Wi-Fi connection, via an on-board router. The mobile device needs to be authenticated to the vehicle to ensure only authorised mobile devices can acquire the vehicle's diagnostic data. If authentication is successful, the mobile device will send a diagnostic command to the ECU, and the mobile will receive the resulting data. The application will interpret the data. This way, the driver is always aware of his/her vehicle's condition. In addition, the consistency of data can be maintained between the mobile application and the vehicle.

Data Logging

Data to be logged in the DiaLOG application are as follows:

(i) DTCs: are the error codes associated with the components in the vehicle. The main function of a diagnostic is to read the DTCs and to resolve the associated problems in the vehicle according to the codes.

(ii) ECU content is the firmware, application and data available in each ECU. To retrieve all the data in all the ECUs would be time consuming and require a large memory to store it all. The (concatenated) hashed value of each ECU can be stored to provide an integrity check. Using the architecture as proposed in the EVITA project [21], the master ECU contains all the hashed values of all the ECUs. Any changes in the content of the ECUs, i.e. any write operation to the flash, will change the hashed value stored in the master ECU. Hence, the master ECU is also alerted to the changes. Changes in the master ECU are reported back to the car manufacturer's server. The DiaLOG application data is also updated accordingly.

(iii) Interface connection: to the vehicle is logged in the DiaLOG application. The authentication process from the interface connection will also be logged by DiaLOG to obtain the identity of the entity involved. The identification, authentication method, interface, time and location, which are related to communication events (for example firmware updates) are among the relevant data to be logged.

(iv) Crash-like data: There are two different ways in which EDRs can record crash incident data. The first option is by continuously recording and overwriting the data on the EDR EEPROM/flash. The second option is by recording only when there is crash-like data. Similar to an EDR, crash-like parameters such as a sudden change in velocity could be a triggering factor for the DiaLOG application to start recording the required parameters for crash incidents. Triggering uses less mobile battery capacity than continuous polling of data. The crash-like data will not be overwritten even if the driver keeps on driving after the accident. Once the storage is triggered, the data will be stored permanently until retrieved.

(v) Change in the frequency of messages through the CAN bus: could be an indicator of a potential remote attack being conducted. DiaLOG will record the normal frequency of messages and compare this to the current operational frequency. For example, as explained earlier, the normal frequency of a sideways acceleration sensor is 35 frames per second.

Storage to Cloud

The user can transfer the data from the mobile device to the cloud. For example, after each driving cycle, all data is transferred to the cloud as a storage backup. At any time, the data can be retrieved and analysed.

During Forensics

Requirements during forensics include availability of mobile application data; availability of ECU data; and authenticity, integrity and correctness of data. The latest diagnostic data stored in the mobile device is read during data collection. The ECU data, including the EDR, is also read. During the forensic analysis process, this data is compared to ensure its consistency.

6.6.3 Protocols

There will be two levels of mobile device authorisation with different data access: basic and full, as shown in Table 6.1. The protocol notation is shown in Table 6.2.

Table 6.1: Credentials for read access

Data	User			
	Car owner	Car rental	Potential buyer	Investigator
DTC	✔	✔	✔	✔
Hash chains	✔	✔	✔	✔
External device	✔	✘	✘	✔
Crash data	✔	✘	✘	✔
Bus attack	✔	✘	✘	✔
Crash history	✔	✔	✔	✔
Authorisation	Full	Basic	Basic	Full

Protocol Description

There are two protocols depending on the access authorisation.

Full authorisation: For a full authorisation, the intended entity (e.g. the car owner) must be registered with the car manufacturer. After installation of the DiaLOG application, a registration process is proposed. By registering, the mobile device, Mo,

Table 6.2: DiaLOG protocol notation

CCU	Central Communication Unit
Mo	Mobile device of car owner (full authorisation)
Mt	Mobile device for temporary access (basic authorisation)
k_{mc}	Symmetric key for CCU (shared between Mo and CCU)
k_{temp}	Symmetric key shared between Mt and Mo
id_{Mo}	Identification number of mobile device of car owner
id_{Mt}	Identification number of mobile device for temporary access
id_{ccu}	Identification number of CCU
k_s	Session key shared between mobile device and CCU
pk_{ccu}	Public key of CCU used to verify signature
sk_{ccu}	Private key of CCU used to sign
n_c, n_{Mo}, n_{Mt}	Nonces
$reqdtc$	Request to access DTC
dtc	Diagnostic Transmission Code
ENC	Encryption using AES128
$sign$	Signature using RSA1024
$=$	Equals
$a\|\|b$	a is concatenated with b

will have access to the car via a set of keys (k_{mc} and pk_{ccu}) provided by the car manufacturer. Car owners and law enforcement are given full authorisation provided they are registered with the car manufacturer. The key is a symmetric key shared between the mobile device and the car, k_{mc}. The CCU is the central unit that interfaces communication of the in-car ECUs with the outside world. The symmetric key, k_{mc}, is a medium-term key that requires an update from the car manufacturer. It will be used to authenticate the mobile device to the car's CCU. As shown in Table 6.3, the mobile device Mo will start the protocol by sending its ID, concatenated with an encrypted message using the pre-shared key k_{mc} containing the ID of the CCU, a request to access the data $fullreq$ and a generated nonce n_{Mo}. The CCU will verify the request and the ID of the CCU, and obtain n_{Mo} by decrypting the message received. It will then reply with its ID, concatenated with an encrypted message using the pre-shared key k_{mc} containing the ID of the CCU, the nonce from Mo from the previous message, n_{Mo} and a session key k_s. After a mutual authentication and freshness verification (Step 1-2), the mobile device will request the DTC from the CCU. This message is encrypted using the session key k_s obtained from the previous message, to provide confidentiality. The encrypted dtc transmitted from the CCU will then be signed to ensure its integrity. The signature is in the format of signature with appendix. The authorised mobile device will be able to decrypt the message to obtain the dtc and also verify that its integrity is protected by verifying the signature.

Table 6.3: Full authorisation data access

1.	$Mo \rightarrow CCU$	:	$id_{Mo}\|\|M1$
			$M1 = ENC_{k_{mc}}\{id_{ccu}\|\|fullreq\|\|n_{Mo}\}$
2.	$CCU \rightarrow Mo$	:	$id_{ccu}\|\|M2$
			$M2 = ENC_{k_{mc}}\{id_{Mo}\|\|k_s\|\|n_{Mo}\}$
3.	$Mo \rightarrow CCU$	:	$id_{Mo}\|\|M3$
			$M3 = ENC_{k_s}\{id_{ccu}\|\|reqdtc\}$
4.	$CCU \rightarrow Mo$	:	$M4\|\|sign_{sk_{ccu}}\{M4\}$
			$M4 = ENC_{k_s}\{id_{ccu}\|\|dtc\}$

Basic authorisation: Entities included in this group include anyone interested in renting a car from a company, or a potential buyer when a car is being resold. In order to be given access authorisation, the person interested must acquire a key from the car owner. The key is transmitted and stored in the mobile device of the interested party. It is then used to authenticate the mobile device to the car. Basic authorisation only gives limited data accessibility as described in Table 6.1. The key, k_s is only valid per transaction, i.e. once communication is disconnected, a new key is required to access the data again.

Prior to the start of the protocol, both mobile devices (Mo and Mt) share a symmetric temporary key, k_{temp} and the public key of the CCU, pk_{ccu}. As illustrated in Table 6.4, the temporary mobile device, Mt, will request access to the car from an authorised mobile device (the car owner's), Mo. Mt will send its ID, concatenated with id_{Mo}, and an encrypted message using the preshared k_{temp} containing the CCU's ID, the request and a nonce, n_{Mt}. The car owner's mobile device will decrypt the message using the pre-shared k_{temp} to obtain the nonce generated by the Mt, n_{Mt}. It will then send a message to notify the CCU about the temporary device's request, which contains its ID, the CCU's ID concatenated with an encrypted message using k_{mc} containing the temporary mobile's ID, the request, and nonces n_{Mo} and n_{Mt}. The CCU will decrypt the message to verify that it is communicating with an authorised mobile device, then acknowledge the request by sharing a session key k_s and a generated nonce n_c to the owner's mobile device. The message is encrypted using k_{mc}. The owner's mobile device will then forward the session key and nonce to Mt by sending a message with its ID and the temporary mobile's ID, concatenated with an encrypted message containing id_{Mt}, id_{ccu}, k_s and all the nonces n_c, n_{Mo}, n_{Mt}. Now, the temporary mobile device can communicate with the CCU using the k_s. It will request the DTC from the CCU. The CCU will verify the freshness of the message and the authenticity of Mt.

Table 6.4: Basic authorisation data access

1.	$Mt \rightarrow Mo$	:	$id_{Mt}\|\|id_{Mo}\|\|M1$
			$M1 = ENC_{k_{temp}}\{id_{ccu}\|\|basicreq\|\|n_{Mt}\}$
2.	$Mo \rightarrow CCU$	:	$id_{Mo}\|\|id_{ccu}\|\|M2$
			$M2 = ENC_{k_{mc}}\{id_{Mt}\|\|basicreq\|\|n_{Mo}\|\|n_{Mt}\}$
3.	$CCU \rightarrow Mo$	:	$id_{ccu}\|\|id_{Mo}\|\|M3$
			$M3 = ENC_{k_{mc}}\{id_{Mt}\|\|k_s\|\|n_{Mo}\|\|n_c\|\|n_{Mt}\}$
4.	$Mo \rightarrow Mt$	:	$id_{Mo}\|\|id_{Mt}\|\|M4$
			$M4 = ENC_{k_{temp}}\{id_{Mt}\|\|id_{ccu}\|\|k_s\|\|n_c\|\|n_{Mo}\|\|n_{Mt}\}$
5.	$Mt \rightarrow CCU$	:	$id_{Mt}\|\|id_{ccu}\|\|M5$
			$M5 = ENC_{k_s}\{id_{Mt}\|\|id_{Mo}\|\|id_{ccu}\|\|n_c\|\|reqdtc\}$
6.	$CCU \rightarrow Mt$	:	$M6\|\|sign_{sk_{ccu}}\{M6\}$
			$M6 = ENC_{k_s}\{id_{ccu}\|\|dtc\}$

The message freshness is checked by the nonce it provided for the session, while the authenticity of M_t is through the k_s. The CCU will reply with the encrypted *dtc* to provide confidentiality plus a signature to ensure its integrity. The authorised mobile device will be able to decrypt the message to obtain the *dtc* and also verify that its integrity is protected by verifying the signature.

6.7 Protocol Analysis

In this section, we analyse the proposed protocol in terms of security and performance.

6.7.1 Informal Analysis

Based on the security requirements in Section 6.5, the proposal addresses them as follows:

Integrity: The DTCs being transmitted and stored are signed by the CCU to ensure that the DTCs are integrity protected.

Authenticity: The communicating parties are authenticated for every protocol transaction. They are given access depending on the level of permission prevailing.

Availability: Data availability is ensured by the updating of data every time the authorised mobile device is connected to the car. The data on the cloud will be automatically updated once a connection is available, or whenever the owner is notified.

Reliability: The reliability of the forensic system is improved by having backup data on the cloud as well as on the mobile phone. If the stored data in any of the three different components does not match with the other sources, it shows that the data might be corrupted.

Privacy: The car owner's privacy and driving-related data is protected. The owner has control over the data.

Based on the threat model in Section 6.4, the proposal addresses the threats as follows:

Denial of service (DoS) attack: The data is always automatically transmitted to the phone and later to the cloud. If data is not updated after a certain time, the owner will be notified. Having a backup copy of the data can ensure that the data is available for retrieval if the person is authorised. If the owner himself is the attacker (denying access to the available data in the mobile phone or cloud), the data can always be accessed (by an authorised investigator) directly through the car's CCU or the EDR. If the investigator has access to the mobile device or the cloud data, then the data is always available for investigation. The data on the EDR, mobile device and cloud should be consistent to ensure no data corruption occurs. If any of the data is not consistent, this could indicate an attack. If a DoS occurs through signal jamming, the data is always available on the EDR, since the CAN bus is protected using the proposed security platform. The mobile application will also alert the car owner if the data has not been stored for any particular driving cycle. Hence, the car owner would be alerted if an attack is being carried out.

Impersonation attack: Data can be retrieved by any entity with the correct authorisation, whether it is a full authorisation or a basic authorisation. Instead of using specialised tools, a mobile application provides easy data access without sacrificing authenticity of the person/tool in use.

Data manipulation attack: The content uploaded on the mobile device and cloud are integrity-protected by the use of signatures by the CCU. Furthermore, since this proposal uses a mobile device, the cloud and the ECU as storage devices, there are three different components to verify the consistency of the data. All three components (ECU, mobile device and cloud) should contain the same data. However, if content is manipulated by injecting malicious data through the CAN bus, all three components would share the same falsified data. Our proposal is based on each ECU having a separate security module, and communication through the CAN bus requires authentication. Therefore, any nodes communicating through the CAN bus are authorised.

Insecurities of mobile device: The mobile device could be malicious, or authorised but used with malicious intentions, or compromised. A malicious mobile device would not be able to get access authorisation to retrieve the data. An authorised mobile device with malicious intentions, or compromised, will be able to retrieve the data, but not corrupt or manipulate it. This is because the data is signed by the CCU, and if the mobile device tries to change the data, it would not be able to provide the correct signature. The DiaLOG application may run on a trusted execution environment such as TrustZone [79] and Intel SGX [80], which could further help to protect confidentiality and integrity by providing a secure trusted execution environment.

6.7.2 Formal Analysis

The proposed protocol is formally analysed using CasperFDR and Scyther tools to verify its correctness. For the full authorisation protocol, the required security requirements include:

1. Confidentiality of the secret keys, k_{mc} and k_s, and the secret nonces, n_{mo}, and n_c
2. Authentication properties, which include:
   - Aliveness between Mo and CCU.
   - Agreement between Mo and CCU of k_s and k_{mc}.
   - Synchronisation between Mo and CCU.

For the basic authorisation protocol, the required security requirements include:

1. Confidentiality of the secret keys, k_{temp}, k_{mc} and k_s, and the secret nonces, n_{Mo}, n_{Mt} and n_c.
2. Authentication properties, which include:
   - Aliveness between Mo and CCU, Mt and CCU, Mo and Mt.
   - Agreement between Mt and CCU of k_s and n_c, agreement between Mo and CCU of k_s and k_{mc}.

- Synchronisation between Mt, Mo and CCU.

In this section, only the full authorisation protocol is discussed using CasperFDR script and basic authorisation protocol is discussed using Scyther script. However, we also verified full and basic authorisation protocols using both tools. The scripts are provided in Appendix A.4 and B.4.

Analysis using CasperFDR For CasperFDR, the security properties verified are secrecy, aliveness and agreement. The confidentiality property verifies the secrecy of the k_{mc}, k_s and n_{Mo}, which are shared between the mobile device (Mo) and the CCU. The aliveness property verifies the aliveness between the mobile device (Mo) and the CCU. The agreement property is to ensure the agreement of k_{mc} and k_s shared between mobile device (Mo) and CCU. The intruder has knowledge of all the entities (CCU and Mo) and the request messages to access the data ($fullreq$ and $reqdtc$).

As shown in Appendix A.4.1, the script starts with #Free variables declaration, which declares all the variables used in the protocol. It is followed by the #Protocol description. This describes the messages being transmitted (in sequence) during the authentication and diagnostics, which start with a request from MD to the CCU (i.e. *1.a->b:a,*{*b,fullreq,nmo*}{*kab*}). In *5.a->b:a,reqdtc,*{*b,reqdtc*}{*ks*}, the MD requests the *dtc* after being authenticated and verifies the freshness in messages 1-4.

In the #Processes, all the entities involved in the protocol and their knowledge are declared. For example *INITIATOR(a,b,kab,nmo,fullreq,reqdtc)*, where *a* is the MD, *b* is the CCU and *nmo* is the random nonce generated by MD. MD knows *kab* which is pre-shared between MD and CCU.

The #Specification declares all the assertions made to verify the security properties. The confidentiality of k_{mc} and k_s are declared as *Secret(a,kab,[a,b])* and *Secret(b,ks,[a,b])*. As an authentication verification, the aliveness property and the agreement property between MD-CCU are verified.

The #Actual variables section describes the names of the actual agents, and the actual variables such as MD and CCU. Nothing is declared in the #Functions section. The #System section again declares all the involved entities in the protocol and their knowledge, but with their actual names. For example, *INITIATOR(MD,CCU,KAB,Nmo, FULLREQ,REQDTC)*. The #Intruder Information declares the intruder X who has knowledge of all the entities involved (MD and CCU) and their corresponding public keys.

All the specifications made were verified and no viable attack was found for any of the assertions.

Analysis using Scyther For Scyther, the security properties verified were non-injective synchronisation, non-injective agreement, weak agreement, aliveness and secrecy. The secrecy property verifies the confidentiality of the k_s (shared between Mt and CCU, k_{mc} (shared between Mo and CCU) and the confidentiality of k_{temp} that is shared between the Mt and Mo. The non-injective synchronisation property is to verify that parties (Mt, Mo and CCU) know who they are communicating with, agree on the content of the messages and the order of the messages. The non-injective agreement verifies that parties agree on the content of the variables (n_{Mo}, n_{Mt}, n_c, k_s, k_{mc} and k_{temp}). The aliveness property verifies that the intended communication partner (Mt-Mo, Mo-CCU and Mt-CCU) has executed some events.

As shown in Appendix B.4.2, the script starts with functions declarations. Then, we have macros of messages to make the script neat and easily followed. Next, the events and claims are made for each role (starts with MT, followed by MO and CCU). For example, for the MT role, the events are *send_1 (mt,mo,m1)*, *recv_4 (mo,mt,m4)*, *send_5 (mt,ccu,m5)* and *recv_6 (ccu,mt,m6)*, which means MT sends the macro $m1$ to MO and later, receives macro $m4$ from MO and sends macro $m5$ to the CCU to then receive macro $m6$ from the CCU. Claims are the security properties to be verified. For example, for the MT role, *claim_R5 (mt, Secret, ks)*, *claim_R6 (mt,Secret, k(mo,mt))*, *claim_R6 (mt,SKR, nmo*, *claim_R6 (mt,SKR, nc* and *claim_R6 (mt,SKR, nmt* are for confidentiality. Authentication properties are verified through Agreement (such as *claim_x2 (mt, Weakagree)*, *claim_x4 (mt, Niagree)*), Synchronisation (*claim_x3 (mt, Nisynch)*), and Aliveness (*claim_x1 (mt, Alive)*).

The default verification setup was used (i.e. five as the maximum number of runs, type matching and finding the best attack with a maximum of ten patterns per claim). The results for all the claims made were verified as “Ok” in the “Status” with “Verified” and “No attacks” in the “Comments”. This means that no attack was found within the bounded or unbounded statespace; the security property was successfully verified [118].

6.7.3 Implementation

The proposed protocol was then implemented on a PIC Microchip microcontroller (PIC32MZ2048ECM144) and an Android device (LG Nexus 5) to obtain indicative performance results.

Implementation Platform

The implementation platform for the mobile device, CCU and ECU were the same as mentioned in Section 5.7.3. The applications for basic and full authorisation protocols on the CCU were 43.4 kB and 42.9 kB respectively.

Programming and Debugging Environment

The same programming and debugging environment mentioned in Section 5.7.3 was used for this implementation.

Experiment Setup

Hardware setup The hardware setup for communication between mobile device, CCU and ECU was the same as mentioned in Section 5.7.3.

Software setup RSA1024 was used for digital signatures and AES128 was used for encryption. Based on the proposed protocol, the length of a message was more than 8 bytes; hence, each message needed to be divided into more than one CAN message due to the limited number of bytes (8 bytes) of data per CAN message transmission.

Implementation Results

The computation and communication performance for full and basic authorisation protocols was as shown in Table 6.5. Communication includes the transfer of data from the Wi-Fi module to the middle interface module (via UART) and from the middle interface module to the CCU (via CAN). The communication time can be further improved if CAN FD [130] is used, where one message can contain up to 64 bytes of data, instead of just 8 bytes.

To the author's knowledge, there is no existing related work on automotive forensics that the current performance can be compared with. However, the total time for the protocol to complete is only about 360 ms for full authorisation, and 434 ms for basic authorisation. Basic authorisation takes a longer time to complete since there are additional messages using the temporary mobile device. Although the computation times for AES and HMAC are faster using PIC32MZ as compared to the Android phone, the RSA computation is longer for PIC32MZ. This is because PIC32MZ has cryptographic engines for AES and HMAC which compute the algorithms at hardware level. Hence, this results in a faster computation time. The communication time is longer for the third part of the protocol because the messages from the mobile device need to go through WiFi, be converted to UART messages, then to CAN messages.

Table 6.5: Full and basic authorisation protocol performance

Protocol	Message	Time(ms)				
		Computation			Communication	Total time
		Mo	Mt	CCU		
Full	1	1.972	-	0.055	57.605	59.632
	2	0.506	-	0.082	57.991	58.579
	3	0.458	-	0.037	42.423	42.918
	4	1.068	-	39.447	157.676	198.391
Total		**4.005**	-	**39.820**	**315.695**	**359.520**
Basic	1	0.724	1.861	-	19.650	22.235
	2	1.956	-	0.059	34.832	36.847
	3	0.660	-	0.149	80.995	81.804
	4	0.922	0.500	-	19.650	21.072
	5	-	0.752	0.041	72.787	73.580
	6	-	0.994	39.738	157.676	198.322
Total		**4.262**	**4.107**	**39.900**	**385.590**	**433.859**

It is the same for the communication from the CCU to the mobile device, where the messages from the CCU are in CAN, then converted to UART, and later to WiFi. The baud rates of communication are 9600 bps for UART and 1 Mbps for CAN. The communication time can be further improved if CAN FD [130] is used, where one message can contain up to 64 bytes of data, instead of just 8 bytes.

6.8 Summary

A secure framework for vehicle forensics is proposed to ensure the security of data and at the same time protect users' privacy. The DiaLOG application proposed uses a new framework of automotive forensics, which provides usability and reliability. Our proposal is based on the new ECU architecture where a security module is included in or part of the ECU. By having a mobile application as a logging platform for the vehicle operation, forensic investigation can be more effective. More data options can be stored, thus increasing the accuracy of forensic analysis.

Chapter 7

Vehicles' Maintenance Services Logging System

Contents

A mobile application as a logging platform may provide an automated logging system for car maintenance services. This system helps car owners to keep track of their car maintenance history. A secure protocol to perform the automated logging can ensure the integrity of the records. The use of a mobile device gives a user interface as well as connectivity for the car.

7.1 Introduction

A maintenance services logging system is a useful tool allowing car owners to keep track of the car's condition and can also increase the market value of the car. Logging systems range from manual, paper-based methods to automated, cloud-based approaches. An automated process provides ease of use and easy availability of the records. However, a secure protocol is required to ensure that both service records and workshops are authentic, and hence provide a reliable record. In this chapter, we propose a secure protocol for automated maintenance services logging systems, through the use of a mobile application called AutoLOG. The multiple Electronic Control Units (ECUs) used to support the connected and intelligent vehicle's technology are used to support the digital automated logging system. The records are stored in an authorised mobile device and uploaded onto a cloud server to ensure availability. The proposed protocol is implemented to measure its performance and formally analysed using CasperFDR and Scyther with no known attacks found.

Chapter organisation The first section, Section 7.2 discusses our motivations for developing a maintenance services logging system. Next, Section 7.3 describes the related work on maintenance services logging systems. The threat model for maintenance services logging systems follows in Section 7.4. This material provides the background for the development of the security requirements as discussed in Section 7.5. Our contribution, which is called the AutoLOG protocol, improves the existing implementation of maintenance services logging systems, and is discussed in Section 7.6. Next, Section 7.7 describes the analysis of the AutoLOG protocol, which includes informal and formal analysis, and our implementation of the protocol.

7.2 Motivations

The challenges in a maintenance services record system are to ensure the integrity, authenticity and reliability of the data. Normally the car owner normally does not have access to the data, unless it is manually recorded in a log book that he/she keeps. The car owner cannot validate the services being provided, and can only trust the information provided by the workshop through receipts or documents provided. Furthermore, it is inconvenient to keep receipts and/or documents for all maintenance services throughout a vehicle's lifespan. Equally, a potential buyer does not have an assurance that the records in the maintenance log, and the workshops who performed the services, are authentic.

7.3 Related Work

There are many mobile applications commercially available that provide maintenance service logging systems [155, 156, 157, 158, 159]. However, these applications require manual information to be input by the car owner. After a car is serviced or repaired, the information can either be keyed in, or a photo captured to be stored. The data can later be stored on the cloud, depending on the application feature. Some applications require a manual upload to the cloud, while others will store the data automatically to the chosen cloud server.

Another type of maintenance service logging system is the type provided by car dealers [160, 161]. When a car is being serviced by a car dealer, details are uploaded onto the car dealer's server. When the next service date is approaching, the car owner will be contacted by the car dealer as a reminder.

A recent development in car maintenance logging systems is "AUTObiography" by Motoriety [162]. This service logs the maintenance services record onto the cloud. Trusted workshops registered with Motoriety can use the service and will digitally sign the services performed to be stored on the cloud. The data or the "biography" of the car will then be available on the cloud, and can be passed from one owner to another. All the records are managed by the service, can be retrieved by the car owners, and owners receive reminders about the next service date.

In the above-mentioned works, trust is mainly given to the workshops. If an untrustworthy workshop fakes an item in the list of services conducted, it cannot be proven. In addition, a trustworthy workshop might mistakenly insert an item as a result of human error, since recording and keying in the data is done manually.

There are proposals for reminder notifications of the next service date [163, 164]. There is also a system proposed using a passive Radio Frequency Identification (RFID) device to detect the repairs/services being conducted [165].

Suresh *et al.* proposed an Android application to perform vehicle diagnostics [166]. The Android application is installed on a mobile device, and is provided by the car manufacturer. The mobile device is then connected to the car via USB or Bluetooth. The data collected by the applications, which includes driving patterns and vehicle condition, is then transmitted to the car manufacturer's cloud server. This information can help the car manufacturer to find any faults arising in the car. The driver will be notified of any fault through voice alert.

Table 7.1 shows the added features of AutoLOG compared to other related systems, which include manual and automated systems. AutoLOG, AUTObiography and a car dealer's cloud server provide automation, but the mobile applications do not

Table 7.1: Features of AutoLOG compared to other related works

Features	AutoLOG	AUTObiography	Mobile applications	Car dealer's cloud server
Automation	✔	✔	✘	✔
Data ownership	✔	✔	✔	✘
Data availability	✔	✔	✔	✔
Validation of services	✔	✘	✘	✘
Security	✔	✔	✘	✔
Options of workshops	✔	✔	✔	✘

[155, 156, 157, 158, 159]. Ownership of the data recorded belongs to the car owner in AutoLOG, AUTObiography and the discussed mobile applications. However, ownership of the records in a car dealer's cloud server belongs to the car dealer. Data availability is supported by all the systems discussed, including AutoLOG. However, since the uploading process is manual for these mobile applications, data availability depends on this manual process. Unlike other related systems, our proposal considers the capability of the ECUs to validate the services. Security is a feature provided by all three automated systems. Car owners have the flexibility to choose from a range of different workshops for AutoLOG, AUTObiography and the discussed mobile applications. However, in the car dealer's cloud server system, workshop options are limited to the ones appointed by the car manufacturers.

7.4 Threat Model

In a maintenance services logging system, assets to be protected are the read and write access authorisation and the authentication and integrity of the data. Potential attackers are untrustworthy workshops, owners and hackers with financial motivation, and potential buyers attempting to reduce the selling price. Two of the most likely threats are:

i) Dishonest mechanic charges owner for a full-service, but may have done little/nothing.

ii) Owner changes service log to make the car more attractive to a buyer.

There are a number of possible attacks that could be performed in a digital maintenance logging system as follows:

Denial of Service (DoS) attack: To cause an availability issue, where data stored is not able to be retrieved, or data cannot be stored. Denying access to data to an authorised party is also a method of DoS.

Impersonation attack: To impersonate an authorised party to conduct further attacks; for example, an attacker impersonating an authorised workshop to log a record showing that the service was conducted by a certain trusted workshop.

Data manipulation attack: To change the list of services, by changing the data either before or after storage.

Replay attack: By replaying the same record of service to be stored on a different date to fake a record. An additional assumption in the threat model in digital automated systems is that the attacker cannot break well-established cryptographic algorithms.

7.5 Security Requirements

Although there are many security requirements for this application, we discuss the main security requirements based on the threat model and the potential risks related to maintenance services logging as discussed in Section 7.4. In general, a maintenance services logging system should satisfy the following security requirements:

Integrity: The data stored should not be able to be changed, modified or added to, in order to ensure that the record of maintenance service is integrity protected.

Authentication: Data authentication and data origin authentication should be in place. This is to ensure that the data comes from an authorised party (workshop and car) and the data itself is authentic.

Non-repudiation: Assurance that the data stored by the workshop can be verified, i.e. the workshop cannot deny that the data stored originated from its diagnostic tool and services were conducted by the workshop or dealer.

Freshness: Assurance that no replay attack is possible. A record of services cannot be logged if it is not actually performed. The records should not be linked to a car owner's Personally Identifiable Information (PII). Hence, privacy is not a concern in the maintenance services record.

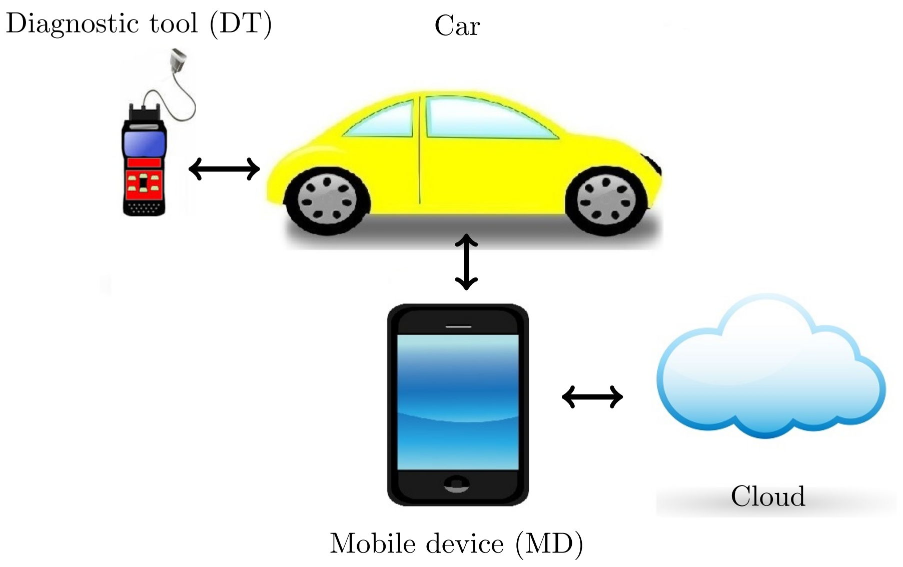

Figure 7.1: Framework for the automated maintenance logging system

7.6 Proposed Solution

The framework is shown in Fig. 7.1. The mobile device provides both a graphical user interface (GUI) and connectivity. The mobile application supporting our proposed protocol is called AutoLOG. The process starts with the workshop updating (on the car) the list of services conducted, the date the service was performed and the date of next service. In order to communicate with the car, the workshop uses a diagnostic tool (DT). The diagnostic tool will communicate with the car through its Central Communication Unit (CCU). The CCU is a type of Electronic Control Unit (ECU). It is the first node any external device will have to go through in order to communicate with other ECUs. The related sensors and/or ECU(s) for the service will validate the information given to the CCU by the DT. After validation, the CCU will store the latest record of maintenance services. The mobile device will then retrieve this data from the CCU and upload the data to the cloud. This way, the records are always available in both the mobile device and the cloud. If the mobile device is lost, the data is still always available on the cloud. In this proposal, the trust foundation is moved from the workshop to the car's sensors and ECU nodes. For the cloud server, there are a few options, including being owned by user; a trusted third party; and a community or government body. Cloud ownership is beyond the scope of this work.

Our proposal is based on the EVITA project [21], which proposed an embedded Hardware Security Module (HSM) in the ECU to ensure secure communications for on-board systems. As proposed in the EVITA project, each ECU has its own HSM.

This suggests that any node communicating through the CAN bus is required to have access authorisation in order to send or receive messages. In our proposal, the mobile device and diagnostic tool act as communicating nodes through the CAN bus, and so require access authorisation.

A car owner has to register his/her mobile device with the car manufacturer/trusted third party appointed by the car manufacturer. Following registration, the mobile device and the car will obtain and share a set of data/keys such as a symmetric key shared between the mobile device and the CCU, the identification number of the mobile device and the identification number of the CCU. In the event of a change in ownership of the car or the mobile device, the car owner is responsible for performing the deregistration. Once registered, the mobile device can retrieve the data whenever maintenance services are performed. The data from the car is bound by the CCU's signature. Verification of data can be performed by decrypting the signature using the public key of the CCU.

The reason for not choosing a Transport Layer Security (TLS) protocol for this application is because it is too much for CCU/ECU devices to cope with. Furthermore, a TLS protocol is bulky and has many implementation options, which can lead to increased vulnerabilities. Our proposed protocol is very specific for this application, eliminating additional vulnerabilities. The TLS protocol is also slower in performance [167].

7.6.1 Protocol Goals

This section discusses the requirements of each party involved in an automated maintenance service log update.

Car: The car requires authentication of the diagnostic tool, authentication of the mobile device and data integrity of the information transferred from the diagnostic tool.

1. CCU: The CCU is the interface for the MD or DT to communicate with other related ECUs.

2. ECU: Some cars are pre-set with ECUs that are able to detect any changes made, such as changing the engine oil. We assume all ECUs will be able to do this in the future.

Mobile device (MD): The mobile device requires authentication of the car (CCU) and data integrity of the information transferred from the CCU.

Diagnostic tool (DT): The diagnostic tool requires authentication of the car (CCU).

7.6.2 Protocol Assumptions and Preconditions

Assumptions and preconditions relating to the successful use of AutoLOG are as follows.

1. The mobile application is installed on a mobile device and the cloud server is properly set up for the data to be stored.
2. The nonces generated (by DT, CCU and MD) should be random and not predictable.
3. The ECUs and sensors are equipped with the capability to validate the services being provided. For example, the sensor can validate the parameters given by the CCU, such as the serial ID of a new component. The proposal [165] uses RFID tags attached to the required devices to track maintenance services activities related to the particular devices. This proposal could be used for this purpose. For a start, the firmware update status could be logged. Cars are now full of electronic modules that may require firmware updates. As part of the normal service, logging the status of all this firmware (which may then trigger updates) could be useful. Consequently, when buying a second hand car, the potential buyer not only knows it had a normal service on a particular date, but also knows whether its IT/electronic systems have been serviced (kept up-to-date).
4. The cloud is securely managed. A user is authenticated to access the cloud server, and only authorised users have access to the data. However, even if an attacker is able to get access to the data in the cloud, the main concern is to protect the integrity of the data, which is provided by our protocol.
5. The data is always automatically transmitted to the phone and later to the cloud. If data is not updated after a certain time, the owner will be notified.

7.6.3 Protocol Key Distribution

Figure 7.2 shows the hierarchy for the key distribution. The hierarchy may be implemented by a specific car manufacturer. A car manufacturer may have a list of trusted workshops and diagnostic tool manufacturers. Each diagnostic tool in a workshop has its own set of public and private keys, one set for signature and one set for encryption. The digital certificates containing the public keys of the diagnostic tools that tie the diagnostic tool to a workshop are available at a diagnostic tool manufacturer server, which is under a trusted third party server. The cars and mobile devices are registered

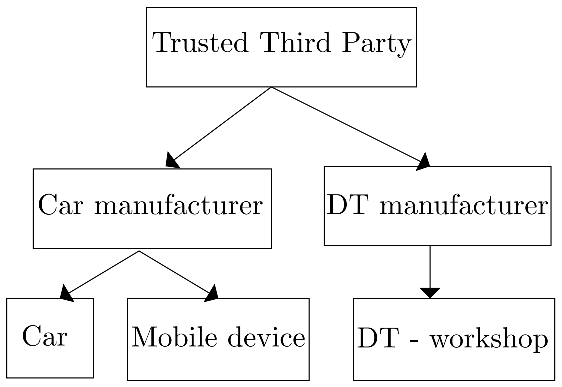

Figure 7.2: Hierarchy for the key distribution

under the car manufacturer, which is also under the same trusted third party server of the diagnostic tool manufacturer. Each CCU has its own set of public and private keys which are pre-installed during manufacturing. These keys are updated by the car manufacturer. The mobile device needs to be registered to the car manufacturer in order to communicate with the car. After the AutoLOG application is installed on the mobile device, registration of the mobile device via the AutoLOG application will enable the mobile device to communicate with the car. Based on the input parameters during registration, which include the Vehicle Identification Number (VIN), the car manufacturer will share a symmetric key, k_{ccu-md} for the CCU with the intended mobile device. Similarly, the diagnostic tool will acquire the public key of the CCU from the trusted third party. The CCU, which is the master ECU in the car, has the records of all ECUs. The records of ECUs include their IDs, the hash content of the firmware and their symmetric keys to communicate with the CCU, $k_{ccu-ecu}$. The keys are stored in the HSM for the car (CCU and ECUs) and the diagnostic tool. For the mobile device, the keys are stored in a secure memory, for example on a secure element.

The data is bound to a car through the car's signature, i.e. the CCU of a car will sign the message. The message can be verified to its origin based on the signature.

There may be issues related to having all devices registered with the car manufacturer, where cryptographic keys need to be loaded; i.e., it may cause control management issues as the devices will need to be properly controlled by car manufacturers. That would mean that any devices not registered with the car manufacturers would not be able to work as they would not be authorised. Car manufacturers will need a proper process for registering both new devices (to be installed during car manufacturing, and also during part/device replacement), and second-hand devices. This would solve the problem of unauthenticated devices being used. We envisage a government agency associated with a road and transport authority (for example the Department for

Transport (DFT) in the United Kingdom) will be the authority providing the trusted third party service.

Table 7.2: AutoLOG protocol notation

DT	Diagnostic tool
CCU	Central Communication Unit
MD	Mobile device
dt	ID of DT
ccu	ID of CCU
md	ID of MD
pk_x	Public key of x, x= DT or CCU
sk_x	Private key of x, x= DT or CCU
na, nb, nc, nd	MAC keys
ne, nf	AES keys
$k_{ccu-ecu}$	Symmetric key shared between CCU and ECU
k_{ccu-md}	Symmetric key shared between CCU and MD
ENC	Encryption using RSA
enc	Encryption using AES128
$sign$	Sign using RSA
MAC	HMAC using SHA256
$mile$	Mileage
$servicetype$	Type of service (basic, full or major)
$servicedate$	Date of service
$nextdate$	Next date of service
$serviceupdate$	Command to conduct the log update
$serviceupdatereq$	Command to obtain the log update
$validateservice$	Command to validate service from CCU to ECU
$serviceupdateready$	Response from CCU to acknowledge it is ready
$ackready$	Response from ECU to acknowledge it is ready
ack	Acknowledgement
$s1, s2, s3$	List of services, repairs or updates conducted
$=$	Equals
$a\|\|b$	a is concatenated with b
$a \oplus b$	a XOR b

Protocol Description

To communicate with the car, the mobile device is connected to the OBD-II port via Wi-Fi or Bluetooth. Once connected, the mobile device is authenticated, to determine whether it is authorised to retrieve the requested data. Once authenticated, the mobile device is connected to the CAN bus and is able to access the required data. The protocol notation is as shown in Table 7.2. The protocol, which is divided into three phases, is shown in Tables 7.3, 7.4 and 7.5. The first part (Table 7.3) shows the communication between the DT and CCU.

1. In the first message, the DT sends its ID, concatenated with an update notification, *serviceupdate*, and a nonce, na. These parameters are signed with the DT's private key and encrypted with the CCU's public key. The digital signature uses signature with message recovery to provide bandwidth efficiency. The signature is then encrypted with the CCU's public key. The objective is to protect the secret nonce na, so that only the authorised CCU is able to obtain the value of na. The CCU decrypts the message and verifies the signature of DT. From that, it obtains the na, to be sent in the second message to the DT.

2. In the second message, the CCU sends its ID concatenated with the acknowledgement receipt of service update command and nonce na. It is concatenated with its own generated nonce, nb. This message is signed with its private key and encrypted with the DT's public key. This signature also uses signature with message recovery and is then encrypted with the DT's public key in order to protect nb.

3. The DT then decrypts the message to get the nonce nb. The nonces na and nb are used for MAC computation for the following messages between DT and CCU. The DT then replies with all the required information, i.e. the type of maintenance service conducted (either basic, full or major), the service date, the next date of service, and the mileage reading, concatenated with the MAC of all the parameters. The MAC is to ensure that the integrity of the data can be verified by the CCU.

4. The CCU acknowledges receipt of these parameters and concatenates acknowledgement with the MAC.

5. Upon receiving the acknowledgement, the DT sends the list of services, repairs or updates conducted, in this example, they are $s1$, $s2$ and $s3$.

6. The next part is the communication between the CCU and the related ECU(s) as shown in Table 7.4. The CCU validates the list of services, repairs and/or updates claimed by the DT. The related ECUs, equipped with sensors to verify the services/repairs/updates conducted, respond accordingly. The CCU sends a command *validateservice* and a nonce nc, which is encrypted with $k_{ccu-ecu}$ to ensure only an authorised ECU can read the nonce.

Table 7.3: DT-CCU update of services protocol

1.	DT $\rightarrow$ CCU	:	$dt \| ENC_{pk_{ccu}}\{sign_{sk_{dt}}\{M1\}\}$ M1= $ccu \| serviceupdate \| na$
2.	CCU $\rightarrow$ DT	:	$ccu \| ENC_{pk_{dt}}\{sign_{sk_{ccu}}\{M2\}\}$ M2=$dt \| ack \| na \| nb$
3.	DT$\rightarrow$ CCU	:	$dt \| M3 \| MAC_{na \| nb}\{M3\}$ M3= $ccu \| servicetype \| servicedate \| nextdate \| mile$
4.	CCU $\rightarrow$ DT	:	$ccu \| M4 \| MAC_{na \| nb}\{M4\}$ M4=$dt \| ack$
5.	DT$\rightarrow$ CCU	:	$dt \| M5 \| MAC_{na \| nb}\{M5\}$ M5=$s1 \| s2 \| s3$

7. The ECU decrypts the message to obtain the nonce *nc*. It then sends a message to acknowledge the receipt of nonce *nc*, advises that it is prepared for the validation process, and sends its own generated nonce *nd*. This message is encrypted with the same $k_{ccu-ecu}$. The CCU then decrypts the message in order to obtain the nonce *nd*. These nonces *nc* and *nd* are used for MAC computation for the following messages between the CCU and the corresponding ECU.

8. The CCU sends the list of services/repairs/updates conducted, concatenated with a MAC.

9. After verifying the MAC received from the CCU, the ECU validates each service/repair through its related sensors. After validating the list, it sends an acknowledgement whether or not the validation is successful, concatenated with a MAC. If all items in the list are true, only the acknowledgement is sent with a MAC. Otherwise, the failed item is included in the message.

10. The last part of the protocol is where the mobile device retrieves the list of services/repairs/updates from the CCU. The mobile device sends a message containing its ID concatenated with a command of *serviceupdatereq* and a nonce *ne*, which is encrypted with a pre-shared symmetric key between mobile device and CCU, $k_{(ccu-md)}$. The encryption is to ensure the confidentiality of the nonce *ne*. Only the authorised CCU is able to decrypt the message and obtain *ne*. This is required because *ne* is to be used as part of a key for AES computation.

11. The CCU decrypts the message to get the nonce *ne* and then replies with a

Table 7.4: CCU-ECU validation of services protocol

6.	CCU $\rightarrow$ ECU	:	$ccu\|\|enc_{k_{ccu-ecu}}\{M6\}$
			M6=$ecu\|\|validateservice\|\|nc$
7.	ECU $\rightarrow$ CCU	:	$ecu\|\|enc_{k_{ccu-ecu}}\{M7\}$
			M7=$ccu\|\|ackready\|\|nc\|\|nd$
8.	CCU$\rightarrow$ ECU	:	$ccu\|\|M8\|\|MAC_{nc\|\|nd}\{M8\}$
			M8=$ecu\|\|s1\|\|s2\|\|s3$
9.	ECU $\rightarrow$ CCU	:	$ecu\|\|M9\|\|MAC_{nc\|\|nd}\{M9\}$
			M9=$ccu\|\|ack$

message stating that a new service is available. If the service has already been retrieved, it sends a different message to inform the MD. The message contains the ID of the mobile device, *serviceupdate* reply, concatenated with nonce ne and its own generated nonce nf. They are encrypted with the pre-shared symmetric key between the mobile device and the CCU, $k_{(ccu-md)}$. The nonces ne and nf are used for AES computation for the messages between the CCU and the MD.

12. The MD then decrypts the message to obtain the nonce nf, and sends an acknowledgement message to the CCU. This message is encrypted using the nonces as the key. Using the nonces as the key can further provide freshness in the key and ensure that every transaction is protected with a different set of keys. Although it is not a good practice to mix cryptographic primitives, the protocol ensures that the nonces are kept confidential throughout the transaction, hence this use is safe.

13. The CCU then starts sending the required service information to the MD, i.e. the type of maintenance service provided (either basic, full or major), the service date and the date of the next scheduled service, the current mileage and the signature of this message. The signature uses signature with appendix. The signature is used to verify that the message originates from the CCU. The record transferred to the mobile device cannot be changed because only the CCU has the private key to sign the message.

14. The MD, upon receiving these data, can verify the origin of the message (i.e. CCU) by verifying the signature. It then acknowledges receipt of this message, in an encrypted message using AES128.

15. The CCU next sends the list of services/repairs/updates conducted. These are

also appended with a signature for the same reason as in step 13, i.e. origin authentication and integrity protection.

16. Finally, the MD, upon receiving and storing these data, sends an acknowledgement encrypted using AES128 to the CCU. This notifies the CCU that the latest maintenance services record has been retrieved.

Table 7.5: MD-CCU request for services update protocol

10.	MD→ CCU	:	$md\|\|enc_{k_{(ccu-md)}}\{M10\}$ M10=$serviceupdatereq\|\|ne$
11.	CCU → MD	:	$ccu\|\|enc_{k_{(ccu-md)}}\{M11\}$ M11=$md\|\|serviceupdate\|\|ne\|\|nf$
12.	MD → CCU	:	$md\|\|ccu\|\|enc_{(ne\oplus nf)}\{M12\}$ M12=ack
13.	CCU → MD	:	$ccu\|\|enc_{(ne\oplus nf)}\{M13\}\|\|sign_{sk_{ccu}}\{enc_{(ne\oplus nf)}\{M13\}\}$ M13=$servicetype\|\|servicedate\|\|nextdate\|\|mile$
14.	MD→ CCU	:	$enc_{(ne\oplus nf)}\{M14\}$ M14=$ccu\|\|ack$
15.	CCU → MD	:	$ccu\|\|enc_{(ne\oplus nf)}\{M15\}\|\|sign_{sk_{ccu}}\{enc_{(ne\oplus nf)}\{M15\}\}$ M15=$s1\|\|s2\|\|s3$
16.	MD → CCU	:	$enc_{(ne\oplus nf)}\{M16\}$ M16=$md\|\|ack$

7.7 Protocol Analysis

In this section, we analyse the proposed protocol in terms of security and performance.

7.7.1 Informal Analysis

Based on the threat model discussed in the previous section, the protocol addresses threats accordingly.

Denial of service attack (DoS): could be conducted:

(i) by stealing the mobile device. If the mobile device is stolen, all the records are still available on the server. A stolen mobile device would not be able to tamper with the available stored data, because the data is signed by the car's CCU.

(ii) by disabling connectivity between the mobile device and the CCU to disable the update. Since the logging process is automated, once a mobile device is authenticated to the CCU, it will ask for an update every time they are connected. If the update is not conducted, the owner will be notified.

(iii) by introducing manual errors. However, the process may repeat and retry the update. The diagnostic tool will likely abort after a few attempts. A notification message will be prompted after a certain retry limit. An error could occur in normal use; however, it could also be evidence of an attack. The data will always be consistent as the mobile device will verify with the CCU whether the last data has been retrieved. If not, the CCU will retain the last record.

(iv) by causing the related ECUs/sensors to malfunction. During the second phase, i.e. the validation of the services, the ECU will acknowledge that the services are being performed correctly as detailed by the DT to the CCU in the previous phase. In this phase, all the related sensors will verify the correctness of the provided data. If any of the sensors fail, this will be displayed on the diagnostic transmission code (DTC, which is the error code) before the services are performed. The faulty sensor should be fixed prior to updating the maintenance services logging system.

Impersonation of recognised workshop or dealer: is prohibited with the use of digital signatures to ensure only an authorised DT can carry out the storing of information to the CCU.

Data manipulation attack: (change, deletion or insertion) could be conducted at three different stages:

(i) From the DT side: A digital signature is used to ensure that only an authorised DT can sign the message required. Therefore, the message is authentic and comes from an authorised party, unless the private key is compromised.

(ii) After storing the information to the CCU, and during retrieval of data from CCU to the MD: The CCU only stores the last record of maintenance service performed. This information is important to the car owner. If an adversary wanted to modify or manipulate this one record, he would need to have access to the CCU information, i.e. the key to read and/or write to the specific memory address.

(iii) After storing the information in the mobile application or server: Fake records could be inserted to increase the car's resale value. With this protocol, this is not possible because the record is validated by the relevant ECU and also protected by the CCU's signature to ensure its integrity. The mileage can also prove the age of the car when the service is conducted.

Replay attack: is not possible through the use of random nonces for each transaction.

The proposal also addresses all the security requirements discussed in Section 7.5 as follows:

Integrity: The data stored cannot be changed, modified or added. To ensure that the record of maintenance service is integrity protected, MAC and digital signatures are used.

Authentication: Data authentication and data origin authentication should be in place. MAC is used to verify the data origin authentication.

Non-repudiation: Digital signatures are used to ensure that the workshop and the car cannot deny their own data.

Freshness: Freshness is verified by using nonces and the mileage reading.

7.7.2 Formal Analysis

The proposed protocol is formally analysed using CasperFDR and Scyther tools to verify its correctness. The protocol is modelled as follows. The DT knows the CCU, but does not know the MD. The MD only communicates with the CCU and not with the DT.

The protocol security objectives are key confidentiality and internal (CCU-ECU) and external (DT-CCU and MD-CCU) authentication. From our CasperFDR and Scyther input scripts, the following security claims are made and verified:

1. Confidentiality of the secret nonces (na, nb, nc and nd: used as the MAC keys, and ne and nf: used as the AES keys), and all secret keys ($k_{ccu-ecu}$ and k_{ccu-md}).

2. Authentication properties, which include:

- Aliveness between DT and CCU, aliveness between CCU and ECU, and aliveness between MD and CCU .
- Agreement between DT and CCU of na and nb, agreement between CCU and ECU of nc and nd, and agreement between MD and CCU of ne and nf.
- Synchronisation between DT and CCU, synchronisation between CCU and ECU, and synchronisation between MD and CCU.

The scripts are divided into three parts for the three different parts of the protocol. In this section, only the first part of the protocol (DT-CCU) is discussed using CasperFDR script and the last part of the protocol (MD-CCU) is discussed using Scyther script. However, we also verified the overall protocol using both tools. The scripts are provided in Appendix A.5 and B.5.

Analysis using CasperFDR The full script for the first part (DT-CCU) is provided in Appendix A.5.1. The security properties verified were confidentiality, aliveness and agreement. The confidentiality property verifies the secrecy of the nonces (na and nb) that are used as keys for MAC computations. The aliveness property verifies the aliveness between DT and CCU. The agreement property ensures the agreement of variables shared between the DT and the CCU (na and nb). The threat model is that the attacker knows all the entities involved, i.e. the DT and CCU, and their corresponding public keys.

The script starts with #Free variables declaration, which declares all the variables used in the protocol. It is followed by the #Protocol description. This describes the messages being transmitted (in sequence) as the information passes from the DT to the CCU, which starts from service update notification (i.e. *1.a ->c:a,*{{*c,serviceupdate,na*}{*SK(a)*}}{*PK(c)*}). In *3. a ->c:a,c,service,mile,h(a,c,service,mile,na,nb)*, the list of services is passed from the DT to the CCU in clear text, but appended with the MAC of the message. The same occurs in *5. a - >c:a,s1,s2,s3,h(s1,s2,s3,na,nb)*. Only the DT and the CCU can compute the MACs and verify them based on the shared keys in the previous message.

In the #Processes, all the entities involved in the protocol, and their knowledge, are declared. For example *INITIATOR(a,c,serviceupdate,na,service,mile,s1,s2,s3) knows PK,SK(a)*, where a is the DT and c is the CCU.

The #Specification declares all the assertions made to verify the security properties. The confidentiality of na and nb are declared as Secret(a,na,[c]) and Secret(c,nb,[c]). As an authentication verification, the aliveness property between DT-CCU and the agreement property between DT-CCU are verified.

The #Actual variables section describes the names of the actual agents, servers and the actual variables, such as agent a is DT and agent c is CCU. In the #Functions section the public and secret keys are declared (symbolic PK,SK). The #System section again declares all the involved entities in the protocol and their knowledge, but with their actual names. For example, *INITIATOR(DT,CCU,Serviceupdate,Na,Service,Mile, S1,S2,S3)*. The #Intruder Information declares an intruder X, who has knowledge of all the entities involved and their public keys, and its own public and secret keys, i.e. IntruderKnowledge=DT,CCU,X,PK.

All the specifications made were verified and no attack was found for any of the assertions.

Analysis using Scyther The full script for the third part of the protocol, i.e. between MD-CCU can be found in Appendix B.5.3. The security properties verified were secrecy, non-injective synchronisation, non-injective agreement and aliveness [117]. The secrecy property verifies the confidentiality of the nonces (ne and nf), which are used as keys for AES computations. The non-injective synchronisation property verifies that all parties (MD and CCU) know who they are communicating with, and agree on the content of the messages and the order of the messages. Non-injective agreement verifies that all parties (MD and CCU) agree on the content of the variables (ne and nf). The aliveness property verifies that the intended communication partner (MD-CCU) has executed some events.

The script starts with functions declarations. Then, we have macros of messages to make the script neat and easily followed. Next, the events and claims are made for each role (MD and CCU).

For example, for the MD role, examples of events are *send_10(md,ccu,m10)* and *recv_11(ccu,md,m11)*, which means MD sends macro $m10$ to CCU and later receives macro $m11$ from CCU. Claims are the security properties to be verified. For example, for the MD role, *claim_I5(md,SKR, ne)* is for confidentiality. Authentication properties are verified through Agreement (*claim_I4 (md,Weakagree)*, *claim_I3(md,Niagree)*), Synchronisation (*claim_I2(md,Nisynch)*), and Aliveness (*claim_I1(md,Alive)*).

The default verification setup was used (i.e. five as the maximum number of runs, type matching and finding the best attack with a maximum of ten patterns per claim). The results for all the claims made were verified as "Ok" in "Status" with "Verified" and "No attacks" in the "Comments". This means that no attack was found within the bounded or unbounded statespace; the security property was successfully verified [118].

Table 7.6: Application sizes on PIC32MZ microcontroller

Protocol part		Code size (kB)		
		DT	CCU	ECU
I	DT-CCU	70.7	70.9	-
II	CCU-ECU	-	63.2	63.0
III	MD-CCU	-	45.0	-

7.7.3 Implementation

The proposed protocol was then implemented on a PIC Microchip microcontroller (PIC32MZ2048ECM144) and an Android device (LG Nexus 5) to obtain indicative performance results.

Implementation Platform

The implementation platform for the mobile device was the same as that mentioned in Section 5.7.3. DT, CCU and ECU used the same platform, which was the same as that mentioned in Section 5.7.3. The application sizes for DT, CCU and ECU are as shown in Table 7.6.

Programming and Debugging Environment

The same programming and debugging environment mentioned in Section 5.7.3 was used for this implementation.

Experiment Setup

Hardware setup The hardware setup for communication between mobile device, DT, CCU and ECU was the same as mentioned in Section 5.7.3.

Software setup RSA1024 was used for digital signatures and public key encryption. A prefixed key of X.509 format was used for its certificate. AES128 was used for symmetric key encryption and HMAC SHA256 was used for MAC. Based on the proposed protocol, the length of a message was more than 8 bytes; hence, each message needed to be divided into more than one CAN message due to the limited number of bytes (8 bytes) of data per CAN message transmission.

Implementation Results

The computation and communication performance was as shown in Table 7.7. The communication included the transfer of data from Wi-Fi module to the middle interface

Table 7.7: AutoLOG protocol performance

Protocol part	Message	Time(ms)			
		Computation		Communication	Total time
		A	B		
I	1	53.041	52.691	1.825	107.557
A=DT, B=CCU	2	52.680	53.012	1.825	107.517
	3	0.102	0.086	0.859	1.046
	4	0.084	0.079	0.752	0.915
	5	0.077	0.086	0.752	0.914
II	6	0.099	0.050	0.537	0.686
A=CCU, B=ECU	7	0.039	0.083	0.537	0.659
	8	0.103	0.083	0.859	1.045
	9	0.083	0.078	0.537	0.697
III	10	0.605	0.049	57.627	58.280
A=MD, B=CCU	11	0.805	0.083	72.818	73.705
	12	0.382	0.036	50.031	50.450
	13	1.216	39.459	163.962	204.636
	14	0.231	0.031	34.841	35.103
	15	1.082	39.609	163.962	204.652
	16	0.199	0.030	34.841	35.069
Total					**882.933**

module (via UART) and from the middle interface module to the CCU (via CAN). To the author's knowledge, there is no related work proposing an automated maintenance services logging system to provide a comparison with the present work. However, the total time for the protocol to complete was only about 883 ms. This shows that the protocol is efficient and practical for implementation. Although the computation time for AES and HMAC was faster using PIC32MZ as compared to the Android phone, the RSA computation is longer for PIC32MZ. This is because PIC32MZ has cryptographic engines for AES and HMAC, which compute the algorithms at hardware level, resulting in a faster computation time. The communication time was longer for the third part of the protocol because the messages from the mobile device needed to go through WiFi, be converted to UART messages, then to CAN messages. It was the same for the communication from the CCU to the mobile device, where the messages from the CCU are in CAN, then converted to UART, and later to WiFi. The baud rates of communication were 9600 bps for UART and 1 Mbps for CAN. The communication time could be further improved if CAN FD [130] is used, where one message can contain up to 64 bytes of data, instead of just 8 bytes.

7.8 Summary

The automated logging of car maintenance services helps car owners to keep track of their car's maintenance record and avoid major breakdowns that can be very expensive. Having a secure protocol to conduct the automated logging ensures that no records can be faked or modified. This will not only help the owner during the ownership of the car, but also during car resale by increasing the car's price by showing that the car has been well maintained. The use of a mobile device gives a user interface as well as connectivity for the car, and thus helps the widespread use of this application since not all cars have connectivity and/or user interfaces.

Chapter 8

Conclusion and Future Work

Contents

This chapter summarises our contributions and discusses directions for future work.

8.1 Summary and Conclusions

This thesis describes research on the security and privacy aspects of vehicular applications in connected cars, with a focus on three selected applications. With the evolution of the internet of things, automotive systems are an important area to be explored, especially in terms of security. The security of automotive systems is particularly important as it involves the safety of vehicular operations. The aim of the thesis was to analyse the current security and privacy aspects of automotive systems implementations in the context of the selected applications. We proposed solutions or improvements to ensure that security and privacy aspects are protected, especially from the point of view of consumers, i.e. the car owners. Since there are many vehicular applications, we choose three applications that represent different phases in the car life cycle. The three applications involve different stakeholders and different processes and thus, they have different requirements and challenges.

The thesis started with a brief introduction to the technical terms relating to connected cars. Automotive systems were introduced by discussing ECUs and in-vehicle communications. Vehicular applications involve the overall automotive ecosystem; the automotive stakeholders were discussed, each with their attack capabilities and motivations. Finally, automotive security issues and requirements were briefly explained.

After providing an understanding of the basics of connected cars, three selected vehicular applications i.e. firmware update, forensics, and maintenance services logging systems, were discussed. The processes, challenges and requirements of each application were discussed. From these discussions, we identified opportunities for improvements. For the first selected application, the vehicular firmware update, we discussed the general requirements for ensuring a successful update. We also discussed the different methods of performing updates of ECUs: conventional updates in workshops or OTA updates. An OTA update can be performed either with the car present in a workshop, or at any convenient place with a specified capability. For the second application, vehicular forensics, the main concerns were the availability of the data during forensic investigations and the privacy of the data from the point of view of the car owner. Full and easy availability of data can improve the accuracy of forensic analysis. In the last application, maintenance service logging systems, a discussion of the current process of logging maintenance services records led us to improve the existing implementation.

In the second part of the thesis, we proposed security and/or reliability improvements for the three selected applications. The proposed improvements not only provided flexibility to the users, but also ensured the security and privacy aspects of the systems. We considered the automotive components under careful control of the car manufacturer to ensure a reliable, safe and secure car. Our proposals are based on the new ECU architecture where a security module is included in the ECU.

For the firmware update application, security is a must to ensure a successful update of a car's ECU. One of the main industrial projects, the EVITA project, proposed a firmware update over-the-air protocol. On examination of EVITA, we found some shortcomings and proposed an improved protocol, EVITA+. Our proposed protocol provided additional assurance by considering both security and general requirements to ensure a successful update. These features can help to provide assurance on the reliability and safety of the car. Our proposed protocol provides additional assurance through the rollback mechanism that protects the firmware confidentiality. It also ensures secure transfer of the flash driver for different memory capabilities of the ECUs. These features can help to provide assurance on the reliability and safety of the car. We presented our threat model and outlined the general and security requirements to ensure a successful firmware update process.

The second contribution was also on the firmware update application, ensuring the distribution of firmware updates is flexible and consumer-friendly. We proposed a secure firmware update protocol for automotive systems using a mobile device. The proposed mobile application ensures the authentication of all parties involved in the update process and the confidentiality of the firmware. Even if the mobile application is compromised, it cannot reveal unencrypted firmware and associated secret/private keys. The different possible architectures for the OTA firmware update were discussed, and we selected the best architecture based on security requirements and flexibility for users.

The third contribution was on the subject of vehicular forensics. After our analysis, we concluded that the accuracy of forensic analysis could be improved with a wider range of parameters for the collected data. Users' privacy also needed to be protected. While ensuring secure data and users' privacy, a secure framework for vehicle forensics was proposed. Having a mobile application as a logging platform for the vehicle operation can make forensic investigations more effective, as more data options can be stored, thus increasing the accuracy of forensic analysis. A secure framework for vehicle forensics is proposed to ensure the security of data and at the same time protect users' privacy. The proposed DiaLOG application used a new framework for automotive forensics, which provides usability and reliability.

Finally, in the automated maintenance services logging system, our main concern was to ensure that records are always available to car owners. This helps car owners to keep track of their car maintenance records and avoid major breakdowns that can be very expensive. We proposed a secure protocol for automated logging to ensure the integrity of the records. It ensures no records can be faked or modified. This will not only help the owner during the ownership of the car, but also when the car is resold, by increasing the car's price by showing that it has been well maintained.

While we were proposing a new protocol for each different application, the introduction of the mobile device in the secure architecture showed that it could further provide flexibility, usability and reliability. The mobile device provides connectivity and a user interface for the car. This is especially useful for older cars that do not have modern infotainment units that can provide connectivity and/or a user interface.

All the proposed protocols were implemented to measure their performance and to show their feasibility. We used a generic microcontroller that has the capability of an actual ECU to emulate the performance of an ECU in implementation. Our proposed protocols were shown to be feasible to implement and showed adequate performance. The applications on the Android device were specifically developed and implemented for the proposed protocol (without other features), and are therefore very small.

In addition to informal analysis, these protocols were also subjected to formal analysis using automated tools, i.e. CasperFDR and Scyther. The tools helped to show that the protocols are secure as they passed all the security requirements as listed in the script specification. From the security analysis and implementation, our proposed protocols were shown to be secure, and feasible to implement.

8.2 Future Work

We wanted to show that while providing flexibility to the users, the security aspects of the applications would not be compromised. As car manufacturers are increasingly aware of the security implications in connected cars, they are more inclined towards using security-enabled platforms. However, they are still in the early stages of adopting security. This is where security experts can play a role in providing a thorough security and risk analysis for each new proposed application.

For a short-term piece of future work, the proposed applications can be implemented in an actual ECU with a security platform, and its performance can be analysed to prove its feasibility. On a mobile device, these applications could be implemented in a trusted execution environment to provide secure execution. The Android applications could be further developed to enhance their features. For example, for the forensic system (AutoLOG), more data could be acquired for logging. As more stakeholders will be interested in these applications and the data provided, security and privacy must not be neglected. The management of data must be properly organised to ensure its protection and ownership.

Key management is another area to be explored. As car manufacturers have the responsibility of ensuring secure and reliable systems, they might want a closed system which is fully controlled by them. For example, they may want to ensure only approved components or service providers are used for their cars. However, end users may prefer more flexible systems where they have more choice. Providing flexibility and options to the users while having a tightly controlled system will be a challenge.

There are many other vehicular applications that can be explored. As different applications may involve different stakeholders, a thorough analysis of the security and privacy implications needs to be performed.

As security implementations are still new in the automotive industry, the choices for security modules would be a good area for exploration. Working to find the optimum security platform for different applications may open up further research opportunities and challenges.

The security of the mobile applications used in automotive systems could be another potential area of research as it could have a high impact on the safety of the vehicular operations. Finding which security solution provides the required properties for designated vehicular applications would be a challenge in itself.

Autonomous vehicle security will be an important area to explore, as autonomous implies that the vehicle is controlled by a system (probably comprising of both software and hardware) which will communicate with the ECUs via the in-vehicle networks. Human intervention will be minimal, hence the system security must be extremely robust to ensure safe independent operation of the vehicle.

Lastly, a major generic area to be explored is the boundary between security, privacy and flexibility, within the complex automotive ecosystem with its many stakeholders.

Appendix A

CasperFDR scripts

Contents

A.1 CasperFDR Tool

The CasperFDR tool analyses Communication Sequential Process (CSP) files using a model checker Failure Divergence Refinement (FDR) [114]. CSP is a notation that shows communicating agents and their corresponding messages in a system. CasperFDR is only supported on Linux OS. FDR2.94 was used for our analysis. There are two ways to use the tool: either use Casper and FDR separately; or use them in one tool which combines the two, called CasperFDR.

Tool and language The input file is a .spdl file written using a text editor in Linux OS. The file is then compiled using the Casper tool, which outputs a .csp file. The .csp file is then verified using the FDR tool. In CasperFDR, the result will show whether any attacks are found using the corresponding specifications. It will also show how an attack could be conducted by an intruder, in a text format.

A.1.1 Input file

The manual for input files is described in [114]. The script starts with the #Free variables declaration, which declares all the variables used in the protocol. It is followed by the #Protocol description. This describes the messages being transmitted (in sequence). In the #Processes, all the entities involved in the protocol, and their knowledge, are declared. The #Specification declares all the assertions made to verify the security properties. The #Actual variables section describes the names of the actual agents, the server and the actual variables. In the #Functions section, all the public and secret keys are declared. The #System section again declares all the involved entities in the protocol and their knowledge, but with their actual names. The #Intruder Information declares an intruder X, with his knowledge.

A.1.2 Security properties

The security properties that can be verified using CasperFDR are confidentiality and authentication properties. The confidentiality property is declared as Secret(a,datax,[a,b]), where a and b are the agents, where $datax$ is to be confidential between them. The authentication properties include aliveness and agreement [116]. The aliveness property is defined by Aliveness(a,b), where from agent a's view, b is alive in the protocol. The agreement property is defined as Agreement(a,b,[datax,datay]), where from agent a's view, he agrees on the $datax$ and $datay$ with agent b during the protocol.

A.1.3 Verification Results

The result will be shown at the bottom of each assertion as “No attack found” or “Attack found”.

A.2 Casper Scripts for EVITA+ Protocol

These scripts are for the protocol described in section 4.7.

A.2.1 EVITA+ Protocol Phase I

```
#Free variables
a,b:Agent
c:Server
s:mainServer
mk,na:Nonce
ack:Data
ts:TimeStamp
PK:Agent->PublicKey
SK:Agent->SecretKey
PK1:Server->PublicKey
SK1:Server->SecretKey

InverseKeys = (PK,SK),(PK1,SK1),(mk,mk),(ssk,ssk),(ksb,ksb),(
    kbc,kbc)
requestseed,request:Message
h:HashFunction
ssk:presharedKey
ksb:presharedKey
kbc:presharedKey


#Protocol description
0. ->a:b
1.a->c:{mk}{PK1(c)},ts,{{mk}{PK1(c)},ts}{SK(a)}
2.c->b:{mk}{kbc},ts,{{mk}{kbc},ts}{SK1(c)}
3.b->c:ack,ts,{ack,ts}{kbc}
4.c->a:ack,ts,{ack,ts}{SK1(c)}
```

```
5.a->b:requestseed ,ts ,h({requestseed ,ts}{mk})
6.b->a:{na}{mk},ts ,h({{na}{mk},ts}{mk})
7.a->b:{h(na,ssk)}{mk},ts ,h({{h(na,ssk)}{mk},ts}{mk})
8.b->a:ack ,ts ,h({ack ,ts}{mk})

#Processes
INITIATOR(a,c,s,mk,requestseed ,ssk ,request) knows PK,PK1,SK(a)
SERVER(c,b,kbc) knows PK,PK1,SK1
RESPONDER(b,a,ack ,na,ssk ,ksb ,kbc) knows PK,PK1,SK(b)

#Specification
Secret(a,mk,[b,c])
Secret(a,na,[b])
Secret(a,ssk ,[b])
Secret(a,SK(a) ,[a])
Secret(b,SK(b) ,[b])
Secret(a,{na}{ssk},[b])
Secret(b,kbc ,[c])
Aliveness(a,b)
Aliveness(a,c)
Agreement(a,c,[mk])
Agreement(b,c,[mk,kbc])

#Actual variables
DT,ECU,X:Agent
CCU:Server
Mk:Nonce
ACKs:Data
TimeStamp=0..0
MaxRunTime=0
RequestSeed ,Request ,SSK:Message
Na:Nonce
KAB,KSB,KBC:presharedKey
InverseKeys = (KAB,KAB) ,(Mk,Mk) ,(Na,Na) ,(KSB,KSB) ,(KBC,KBC)
OEM:mainServer

#Functions
```

```
symbolic PK,SK,PK1,SK1

#System
INITIATOR(DT,CCU,OEM,Mk, RequestSeed ,KAB, Request)
SERVER(CCU,ECU,KBC)
RESPONDER(ECU,DT,ACKs,Na,KAB,KSB,KBC)

#Intruder Information
Intruder=X
IntruderKnowledge={DT,CCU,ECU,X,PK,PK1,SK(X)}
```

A.2.2 EVITA+ Protocol Phase II

```
#Free variables
a,b:Agent
c:Server
s:OemServer
mk:SessionKey
requestexit ,request ,ack:Message
ssk:Nonce
PK:Agent->PublicKey
SK:Agent->SecretKey
PK1:OemServer->PublicKey
SK1:OemServer->SecretKey
PK2:Server->PublicKey
SK2:Server->SecretKey
InverseKeys = (PK,SK),(PK1,SK1),(PK2,SK2),(mk,mk),(ksb,ksb),(
    ssk,ssk)
h:HashFunction
ts:TimeStamp
ksb:presharedKey
Fdr,m1,m2:Data

#Protocol description
0. ->a:b
1.a->s:request,ts,{request,ts}{SK(a)}
2.s->a:{ssk}{ksb}%fek,ts,{{ssk}{ksb}%fek,ts}{SK1(s)}
3.s->a:{m2}{ssk}%frm2,ts,{{m2}{ssk}%frm2,ts}{SK1(s)}
```

```
4.a->b:Fdr,ts,h({Fdr,ts}{mk})
5.b->a:ack,ts,h({ack,ts}{mk})
6.a->b:fek%{ssk}{ksb},ts,h({fek%{ssk}{ksb},ts}{mk})
7.b->a:ack,ts,h({ack,ts}{mk})
8.b->c:{m1}{ssk}%frm1,ts,h({{m1}{ssk}%frm1,ts}{mk})
9.b->c:requestexit,ts,h({requestexit,ts}{mk})
10.a->c:frm2%{m2}{ssk},ts,h({frm2%{m2}{ssk},ts}{mk})
11.a->c:requestexit,ts,h({requestexit,ts}{mk})
12.c->b:{m2}{ssk},ts,h({{m2}{ssk},ts}{mk})
13.c->b:requestexit,ts,h({requestexit,ts}{mk})
14.b->a:ack,ts,h({ack,ts}{mk})


#Processes
INITIATOR(a,s,c,request,mk,requestexit,Fdr) knows PK,SK(a),PK1
RESPONDER(b,a,c,ack,mk,ksb,ssk,m1,requestexit) knows PK,SK(b),
    PK1
OemSERVER(s,a,ssk,ksb,m2,b) knows PK,PK1,SK1
SERVER(c,a,b,ssk,m2,requestexit,mk) knows PK,PK1,PK2,SK2

#Specification
 Secret(a,mk,[b])
 Secret(b,m1,[c])
 Secret(s,m2,[b])
 Secret(a,SK(a),[a])
 Secret(b,SK(b),[b])
 Secret(s,ssk,[b])
 Aliveness(a,s)
 Aliveness(a,c)
 Aliveness(b,c)
 Agreement(a,c,[mk])
 Agreement(b,c,[mk])

#Actual variables
DT,ECU,X:Agent
OEM:OemServer
CCU:Server
```

```
Mk:SessionKey
SSK:Nonce
Req,ACK,ReqExit:Message
TimeStamp=0..0
MaxRunTime=0
KSB:presharedKey
InverseKeys=(Mk,Mk),(KSB,KSB),(SSK,SSK)
FDR,FrmA,FrmB,M2,M1:Data

#Functions
symbolic PK,SK,PK1,SK1,PK2,SK2

#System
INITIATOR(DT,OEM,CCU,Req,Mk,ReqExit,FDR)
RESPONDER(ECU,DT,CCU,ACK,Mk,KSB,SSK,M1,ReqExit)
OemSERVER(OEM,DT,SSK,KSB,M2,ECU)
SERVER(CCU,DT,ECU,SSK,FrmB,ReqExit,Mk)

#Intruder Information
Intruder=X
IntruderKnowledge={DT,CCU,ECU,OEM,X,PK,SK(X),PK1,PK2}
```

A.3 Casper Scripts for FOTA Protocol

These scripts are for the protocol described in section 5.6.

A.3.1 FOTA Protocol Phase I and II

```

#Free variables
a,b,c:Agent
s:Server
knb,kna:Nonce
ack:Data
ts:TimeStamp
PK:Agent->PublicKey
SK:Agent->SecretKey
PK1:Server->PublicKey
```

```
SK1:Server->SecretKey
frmold,frmnew:Data
pskfek:sharedKey
InverseKeys = (PK,SK),(PK1,SK1),(kna,kna),(pskecu,pskecu),(pskfek,pskfek),(knb,knb)
requestdl,requesttransferexit:Message
h:HashFunction
pskecu,pskunlock:presharedKey
idecu:Data

#Protocol description
0. ->a:b
1.a->s:a,{kna}{PK1(s)},ts,{a,{kna}{PK1(s)},ts}{SK(a)}
2.s->a:s,ack,ts,{s,ack,ts}{SK1(s)}
3.a->s:a,requestdl,idecu,ts,h({a,requestdl,idecu,ts}{kna})
4.s->a:s,{pskunlock}{pskecu}%ssk,ts,h({s,{pskunlock}{pskecu}%ssk,ts}{kna})
5.a->s:a,ack,ts,h({a,ack,ts}{kna})
6.s->a:s,{frmnew}{pskfek}%frm2,ts,h({s,{frmnew}{pskfek}%frm2,ts}{kna})
7.a->s:a,ack,ts,h({a,ack,ts}{kna})
8.a->c:{a,knb}{PK(c)},ts,{{a,knb}{PK(c)},ts}{SK(a)}
9.c->b:c,{knb}{PK(b)},ts,{c,{knb}{PK(b)},ts}{SK(c)}
10.b->c:b,ack,ts,{b,ack,ts}{SK(b)}
11.c->a:c,ack,ts,{c,ack,ts}{SK(c)}
12.a->b:a,ssk%{pskunlock}{pskecu},ts,h({a,ssk%{pskunlock}{pskecu},ts}{knb})
13.b->a:b,ack,ts,h({b,ack,ts}{knb})
14.b->a:b,{frmold}{pskfek}%frm1,ts,h({b,{frmold}{pskfek}%frm1,ts}{knb})
15.a->b:a,ack,ts,h({a,ack,ts}{knb})
16.a->b:a,frm2%{frmnew}{pskfek},ts,h({a,frm2%{frmnew}{pskfek},ts}{knb})
17.b->a:b,ack,ts,h({b,ack,ts}{knb})

#Processes
```

```
INITIATOR(a,s,b,c,idecu,kna,requestdl,ack,knb,
    requesttransferexit) knows PK,PK1,SK(a)
SERVER(s,a,b,c,idecu,pskecu,pskfek,pskunlock,frmnew,ack,
    requesttransferexit) knows PK,PK1,SK1
RESPONDER(b,a,s,c,idecu,pskunlock,pskecu,ack,pskfek,
    requesttransferexit,frmold) knows PK,PK1,SK(b)
SERVER2(c,a,b,s,idecu) knows PK,PK1,SK(c)

#Specification
 Secret(a,kna,[a,s])
 Secret(a,knb,[a,b])
 Secret(s,frmnew,[s,b])
 Secret(b,frmold,[b])
 Secret(s,pskfek,[s,b])
 Secret(s,pskunlock,[s,b])
 Secret(s,pskecu,[s,b])
 Aliveness(a,b)
 Aliveness(s,a)
 Agreement(a,b,[knb])
 Agreement(s,a,[kna])
 Agreement(s,b,[pskecu,pskfek,pskunlock])

#Actual variables
MD,ECU,CCU,X:Agent
OEM:Server
KNb,KNa:Nonce
ACKs:Data
 TimeStamp=0..0
 MaxRunTime=0
 RequestDL,RequestExit:Message
FRMOLD,FRMNEW:Data
PSKECU,PSKUNLOCK:presharedKey
PSKFEK:sharedKey
IDECU:Data
 InverseKeys = (PSKECU,PSKECU),(KNa,KNa),(PSKFEK,PSKFEK),(
    PSKUNLOCK,PSKUNLOCK),(KNb,KNb)

```

```
#Functions
symbolic PK,SK,PK1,SK1

#System
INITIATOR(MD,OEM,ECU,CCU,IDECU,KNa,RequestDL,ACKs,KNb,
    RequestExit)
SERVER(OEM,MD,ECU,CCU,IDECU,PSKECU,PSKFEK,PSKUNLOCK,FRMNEW,
    ACKs,RequestExit)
RESPONDER(ECU,MD,OEM,CCU,IDECU,PSKUNLOCK,PSKECU,ACKs,PSKFEK,
    RequestExit,FRMOLD)
SERVER2(CCU,MD,ECU,OEM,IDECU)

#Intruder Information
Intruder=X
IntruderKnowledge={MD,OEM,CCU,ECU,X,PK,PK1,SK(X)}
```

A.3.2 FOTA Protocol Replacement Phase

```
#Free variables
a,b,c:Agent
s:Server
ack,idb,idb1:Data
ts:TimeStamp
PK:Agent->PublicKey
SK:Agent->SecretKey
PK1:Server->PublicKey
SK1:Server->SecretKey
InverseKeys = (PK,SK),(PK1,SK1),(pskecu,pskecu)
requestid,requestverifyid:Message
h:HashFunction
pskecu:Key
rnd:Nonce

#Protocol description
0. ->a:c
1.a->c:a,requestid,ts
2.c->b:c,requestid,ts
3.b->c:b,idb1,ts
```

```
4.c->a:c,idb1,ts
5.a->s:a,requestverifyid,idb1,ts,{a,requestverifyid,idb1,ts}{
    SK(a)}
[idb1==idb]
6.s->a:s,({rnd,pskecu,PK(b)}{PK(c)})%v1,ts,{s,({rnd,pskecu,PK(
    b)}{PK(c)})%v1,ts}{SK1(s)}
7.a->c:a,v1%({rnd,pskecu,PK(b)}{PK(c)}),ts,{s,v1%({rnd,pskecu,
    PK(b)}{PK(c)}),ts}{SK(a)}
8.c->b:c,rnd,PK(c),ts
9.b->c:b,{{rnd}{pskecu}}{PK(c)},ts
10.c->a:c,ack,ts,{c,ack,ts}{SK(c)}

#Processes
INITIATOR(a,s,b,c,requestid,requestverifyid) knows PK,PK1,SK(a
    )
SERVER(s,a,b,c,ack,idb,pskecu,rnd) knows PK,PK1,SK1
RESPONDER(b,a,s,c,ack,idb1,pskecu) knows PK,PK1,SK(b)
SERVER2(c,a,b,s,ack) knows PK,PK1,SK(c)

#Specification
 Secret(a,SK(a),[a])
 Secret(b,SK(b),[b])
 Secret(c,SK(c),[c])
 Secret(s,SK1(s),[s])
 Secret(b,pskecu,[s,b,c])
 Aliveness(a,b)
 Aliveness(a,c)
 Aliveness(s,a)
 Agreement(s,c,[pskecu])


#Actual variables
MD,ECU,CCU,X:Agent
OEM:Server
ACKs,ID:Data
TimeStamp=0..0
MaxRunTime=0
```

```
RequestID , RequestVerifyID : Message
PSKECU: Key
RND: Nonce
InverseKeys = (PSKECU,PSKECU)

#Functions
symbolic PK,SK,PK1,SK1

#System
INITIATOR(MD,OEM,ECU,CCU, RequestID , RequestVerifyID )
SERVER(OEM,MD,ECU,CCU,ACKs,ID ,PSKECU,RND)
RESPONDER(ECU,MD,OEM,CCU,ACKs,ID ,PSKECU)
SERVER2(CCU,MD,ECU,OEM,ACKs)

#Intruder Information
Intruder=X
IntruderKnowledge={MD,OEM,CCU,ECU,X,PK,PK1,SK(X)}
```

A.4 Casper Scripts for DiaLOG Protocol

These scripts are for the protocol described in section 6.6.

A.4.1 DiaLOG Full Authorisation Protocol

```
#Free variables
a,b:Agent
nmo:Nonce
ack:Data
ts:TimeStamp
dtc:Data
PK:Agent->PublicKey
SK:Agent->SecretKey
ks:sharedKey
kab:presharedKey
InverseKeys = (PK,SK),(kab,kab),(ks,ks)
h:HashFunction
fullreq ,reqdtc:Message

```

```
#Protocol description
0. ->a:b
1.a->b:a,{b,fullreq ,nmo}{kab}
2.b->a:b,{a,ks,nmo}{kab}
3.a->b:a,reqdtc,{b,reqdtc}{ks}
4.b->a:{b,dtc}{ks},{{b,dtc}{ks}}{SK(b)}

#Processes
INITIATOR(a,b,kab,nmo,fullreq ,reqdtc) knows PK,SK(a)
RESPONDER(b,a,kab,dtc,ks) knows PK,SK(b)

#Specification
Secret(a,kab,[a,b])
Secret(b,ks,[a,b])
Secret(a,nmo,[a,b])
Aliveness(a,b)
Aliveness(b,a)
Agreement(a,b,[ks,kab])
Agreement(b,a,[ks,kab])

#Actual variables
MD,CCU,X:Agent
Nmo:Nonce
ACKs:Data
TimeStamp=0..0
MaxRunTime=0
DTC:Data
KAB:presharedKey
KS:sharedKey
FULLREQ,REQDTC:Message
InverseKeys = (KAB,KAB),(KS,KS)

#Functions
symbolic PK,SK

#System
INITIATOR(MD,CCU,KAB,Nmo,FULLREQ,REQDTC)
```

```
RESPONDER(CCU,MD,KAB,DTC,KS)

#Intruder Information
Intruder=X
IntruderKnowledge={MD,CCU,X,FULLREQ,REQDTC,PK,SK(X)}
```

A.4.2 DiaLOG Basic Authorisation Protocol

```
#Free variables
a,b,c:Agent
nmo,nc,nmt:Nonce
dtc:Data
PK:Agent->PublicKey
SK:Agent->SecretKey
ks:sharedKey
kac,kbc:presharedKey
InverseKeys = (PK,SK),(kac,kac),(ks,ks),(kbc,kbc)
basicreq,reqdtc:Message

#Protocol description
0. ->a:b
1.a->c:a,c,{b,basicreq,nmt}{kac}
2.c->b:c,b,{a,basicreq,nmt,nmo}{kbc}
3.b->c:b,c,{a,ks,nmo,nc,nmt}{kbc}
4.c->a:c,a,{a,b,ks,nmo,nc,nmt}{kac}
5.a->b:a,b,{a,c,b,nc,reqdtc}{ks}
6.b->a:{b,dtc}{ks},{{b,dtc}{ks}}{SK(b)}

#Processes
INITIATOR(a,b,c,kac,nmt,basicreq,reqdtc) knows PK,SK(a)
RESPONDER(b,a,c,kbc,ks,nc,dtc) knows PK,SK(b)
RESPONDER1(c,a,b,kbc,kac,nmo) knows PK,SK(c)

#Specification
Secret(a,kac,[a,c])
Secret(b,kbc,[b,c])
Secret(b,ks,[a,b])
Aliveness(a,b)
Aliveness(b,a)
Aliveness(a,c)
Aliveness(c,a)
Aliveness(c,b)
Aliveness(b,c)
Agreement(a,b,[ks,nc])
Agreement(b,a,[ks,nc])

#Actual variables
MO,CCU,MT,X:Agent
NMO,NC,NMT:Nonce
KS:sharedKey
KAC,KBC:presharedKey
BASICREQ,REQDTC:Message
InverseKeys = (KAC,KAC),(KS,KS),(KBC,KBC)
DTC:Data

#Functions
symbolic PK,SK

#System
INITIATOR(MT,CCU,MO,KAC,NMT,BASICREQ,REQDTC)
RESPONDER(CCU,MT,MO,KBC,KS,NC,DTC)
RESPONDER1(MO,MT,CCU,KBC,KAC,NMO)

#Intruder Information
Intruder=X
IntruderKnowledge={MT,CCU,MO,X,BASICREQ,PK,SK(X)}
```

A.5 Casper Scripts for AutoLOG Protocol

These scripts are for the protocol described in section 7.6.

A.5.1 AutoLOG Protocol Phase I

```

#Free variables
a,c:Agent
```

```
na,nb:Nonce
PK:Agent->PublicKey
SK:Agent->SecretKey
InverseKeys = (PK,SK),(na,na),(nb,nb)
serviceupdate,reqservicetype:Message
h:HashFunction
ack:Message
service,mile:Data
s1,s2,s3:Data

#Protocol description
0. ->a:c
1.a->c:a,{{c,serviceupdate,na}{SK(a)}}{PK(c)}
2.c->a:c,{{a,ack,na,nb}{SK(c)}}{PK(a)}
3.a->c:a,c,service,mile,h(a,c,service,mile,na,nb)
4.c->a:c,a,ack,h(a,ack,na,nb)
5.a->c:a,s1,s2,s3,h(s1,s2,s3,na,nb)

#Processes
INITIATOR(a,c,serviceupdate,na,service,mile,s1,s2,s3) knows PK
    ,SK(a)
RESPONDER(c,a,nb,ack) knows PK,SK(c)

#Specification
Secret(a,na,[c])
Secret(c,nb,[c])
Aliveness(a,c)
Aliveness(c,a)
Agreement(a,c,[na,nb])
Agreement(c,a,[na,nb])

#Actual variables
DT,CCU,X:Agent
Na,Nb:Nonce
InverseKeys = (Na,Na),(Nb,Nb)
Serviceupdate:Message
Ack:Message
```

```
Service,Mile:Data
S1,S2,S3:Data

#Functions
symbolic PK,SK

#System
INITIATOR(DT,CCU,Serviceupdate,Na,Service,Mile,S1,S2,S3)
RESPONDER(CCU,DT,Nb,Ack)

#Intruder Information
Intruder=X
IntruderKnowledge={DT,CCU,X,PK}
```

A.5.2 AutoLOG Protocol Phase II

```
#Free variables
c,d:Agent
nc,nd:Nonce
kcd:presharedKey
InverseKeys =(kcd,kcd)
h:HashFunction
validateservice,ackready,ack:Message
s1,s2,s3:Data

#Protocol description
0. ->c:d
1.c->d:c,{d,validateservice,nc}{kcd}
2.d->c:d,{c,ackready,nc,nd}{kcd}
3.c->d:c,d,s1,s2,s3,h(d,s1,s2,s3,nc,nd)
4.d->c:d,c,ack,h(c,ack,nc,nd)

#Processes
INITIATOR(c,d,s1,s2,s3,nc,kcd,validateservice)
RESPONDER(d,c,nd,ack,kcd,ackready)

#Specification
Secret(c,nc,[d])
Secret(d,nd,[c])
Secret(c,kcd,[d])
Secret(d,kcd,[c])
Aliveness(c,d)
Aliveness(d,c)
Agreement(c,d,[nc,nd])
Agreement(d,c,[nc,nd])

#Actual variables
CCU,ECU,X:Agent
Nc,Nd:Nonce
KCD:presharedKey
InverseKeys = (KCD,KCD)
ValidateService,AckReady,Ack:Message
S1,S2,S3:Data

#Functions

#System
INITIATOR(CCU,ECU,S1,S2,S3,Nc,KCD,ValidateService)
RESPONDER(ECU,CCU,Nd,Ack,KCD,AckReady)

#Intruder Information
Intruder=X
IntruderKnowledge={CCU,ECU,X}
```

A.5.3 AutoLOG Protocol Phase III

```
#Free variables
ne,nf:Nonce
b,c:Agent
PK:Agent->PublicKey
SK:Agent->SecretKey
kbc:presharedKey
InverseKeys = (PK,SK),(kbc,kbc),(sk1,sk1)
serviceupdateready,serviceupdatereq:Message
sk1:Nonce x Nonce -> sessionKey
ack:Message
```

```
service,mile,s1,s2,s3:Data

#Protocol description
0. ->b:c
1.b->c:b,{serviceupdatereq,ne}{kbc}
2.c->b:c,{b,serviceupdateready,ne,nf}{kbc}
3.b->c:b,c,{ack}{sk1(ne,nf)}
4.c->b:c,{service,mile}{sk1(ne,nf)},{{service,mile}{sk1(ne,nf)
    }}{SK(c)}
5.b->c:{b,ack}{sk1(ne,nf)}
6.c->b:c,{s1,s2,s3}{sk1(ne,nf)},{{s1,s2,s3}{sk1(ne,nf)}}{SK(c)
    }
7.b->c:{b,ack}{sk1(ne,nf)}

#Processes
RESPONDER(c,b,nf,serviceupdateready,ack,kbc,service,mile,s1,s2
    ,s3) knows PK,SK(c),sk1(ne,nf)
INITIATOR(b,c,ne,serviceupdatereq,kbc,ack) knows PK,SK(b),sk1(
    ne,nf)

#Specification
Secret(b,ne,[c])
Secret(c,nf,[b])
Aliveness(c,b)
Agreement(c,b,[ne,nf])

#Actual variables
MD,CCU,X:Agent
Ne,Nf:Nonce
KBC:presharedKey
InverseKeys = (KBC,KBC)
ServiceUpdateReq,ServiceUpdateReady:Message
Ack:Message
Service,Mile:Data
S1,S2,S3:Data

#Functions
```

```
symbolic sk1,PK,SK

#System
RESPONDER(CCU,MD,Nf,ServiceUpdateReady,Ack,KBC,Service,Mile,S1
    ,S2,S3)
INITIATOR(MD,CCU,Ne,ServiceUpdateReq,KBC,Ack)

#Intruder Information
Intruder=X
IntruderKnowledge={MD,CCU,X,PK}
```

Appendix B

Scyther scripts

Contents

B.1 Scyther Tool

Scyther performs an automatic analysis of security protocols in a Dolev-Yao style model, for an unbounded number of instances [115].

B.1.1 Tool and language

Scyther is easily installed on a 64 bit Windows 8.1 Pro machine. The input file for the tool is a .spdl file. There are three components on the menu bar, which are "File", "Verify" and "Help". There are two tabs, i.e. protocol description and setting tabs. In the verify menu, the user can choose to verify the protocol based on the security properties defined in the input file using "Verify protocol", or to check whether the roles can complete the protocol using "Characterise role", or to verify all claims, where the security properties are automatically generated by the tool using "Verify automatic claims". There are two main tabs, i.e. "Protocol description" and "Setting". The "Protocol description" tab is to write in the input file. The "Setting" tab is to set the verification parameters which include the maximum number of runs, and matching type as type matching, find basic type flaw or find all type flaws. The default setting is five runs and type matching. The advanced parameters are search pruning (find all attacks, best attacks or first attacks) and a maximum number of patterns per claim. The font size for the graph can also be set in the setting tab.

After the input file is written, "characterise role" is chosen. After ensuring that the agents in the protocol can complete the protocol through the "reachable" status, the "verify protocol" is chosen. An output window will pop out once the result is obtained for all the claims made.

B.1.2 Input file

The script starts with function declarations. Then, we have macros of messages to make the script neat and easily followed. Next, the events and claims are created for each role. The roles are the agents involved in the protocol. Claims are the security properties to be verified. The claims are based on the agents' local view of the state of the system. The protocol should ensure that some properties of the global state of the system can be known to the agents based on this local view.

B.1.3 Security properties

In the Scyther tool, the authentication properties are verified through agreement, aliveness and synchronisation properties. Aliveness verifies the authentication of communication partners. Synchronisation is a stronger authentication requirement that verifies not only the communication partners, but also the sequence of exchanged messages [119]. Agreement verifies the data exchanged between the agents [117].

For Scyther, the security properties verified are non-injective synchronisation, non-injective agreement, weak agreement, aliveness and secrecy. The default verification setup was used (i.e. five as the maximum number of runs, type matching and finding the best attack with a maximum of ten patterns per claim). The secrecy property verifies the confidentiality of the secret keys or data. The non-injective synchronisation property verifies that parties know who they are communicating with, and agree on the content of the messages and the order of the messages. The non-injective agreement verifies that parties agree on the content of the variables. In Scyther, all the security properties are modelled as role-based. Each entity is considered as one role. The properties are viewed from the local view of each role.

B.1.4 Verification results

The default verification setup was used (i.e. five as the maximum number of runs, type matching and finding the best attack with a maximum of ten patterns per claim). If the results for all the claims made are verified as “Ok” in the “Status” and “No attacks within bounds” in the “Comments”, this means that no attack was found within the bounded statespace, but there may be an attack possible outside the bounded statespace [118]. Otherwise, if the results for all the claims made are verified as “Ok” in the “Status” with “Verified” and “No attacks” in the “Comments”, this means that no attack was found within the bounded or unbounded statespace; the security property has been successfully verified [118].

The results can also be falsified, which means there can be at least one attack possible on the protocol. A potential attack is shown graphically in a separate window if the user clicks the button to see how the attack is possibly conducted. The user can see the potential attack in arrows and boxes representing events and claims.

B.2 Scyther Scripts for EVITA+ Protocol

These scripts are for the protocol described in section 4.7.

B.2.1 EVITA+ Protocol Phase I

```
hashfunction h;
function f;
```

```
usertype Timestamp;

protocol evita1(dt,ccu,ecu){
const SSK;
const pskC;

macro m1 = {Mk}pk(ccu),ts,{{Mk}pk(ccu),ts}sk(dt);
macro m2 = {Mk}k(ccu,ecu),ts,{{Mk}k(ccu,ecu),ts}sk(ccu);
macro m3 = ack,ts,{ack,ts}k(ecu,ccu);
macro m4 = ack,ts,{ack,ts}sk(ccu);
macro m5 = Requestseed,ts,h({Requestseed,ts}Mk);
macro m6 = {Na}Mk,ts,h({{Na}Mk,ts}Mk);
macro smk = f(ssk,Na);
macro m7 = {smk}Mk,ts,h({{smk}Mk,ts}Mk);
macro m8 = ack,ts,h({ack,ts}Mk);

role dt{
fresh Mk:Nonce;
var ack: Data;
fresh ts:Timestamp;
const Requestseed: Data;
var Na:Nonce;
const ssk:Data;

send_1(dt,ccu,m1);
recv_4(ccu,dt,m4);
send_5(dt,ecu,m5);
recv_6(ecu,dt,m6);
send_7(dt,ecu,m7);
recv_8(ecu,dt,m8);

claim_a1(dt,Alive);
claim_a2(dt,Weakagree);
claim_a3(dt,Niagree);
claim_a4(dt,Nisynch);
claim_a5(dt,SKR,Mk);
claim_a6(dt,Secret,sk(dt));
```

```
claim_a7(dt,Secret,Na);
claim_a8(dt,Secret,ssk);
claim_a9(dt,Secret,smk);
}

role ccu{
var Mk:Nonce;
var ack: Data;
fresh ts:Timestamp;
var FrmA;
var FrmB;
var requesttransferexit:Data;

recv_1(dt,ccu,m1);
send_2(ccu,ecu,m2);
recv_3(ecu,ccu,m3);
send_4(ccu,dt,m4);

claim_b1(ccu,Alive);
claim_b2(ccu,Weakagree);
claim_b3(ccu,Niagree);
claim_b4(ccu,Nisynch);
claim_b5(ccu,SKR,Mk);
claim_b6(ccu,Secret,sk(ccu));
}

role ecu{
var Mk:Nonce;
const ack: Data;
fresh ts:Timestamp;
fresh Na:Nonce;
var Requestseed: Data;
const ssk: Data;

recv_2(ccu,ecu,m2);
send_3(ecu,ccu,m3);
recv_5(dt,ecu,m5);
```

```
send_6(ecu,dt,m6);
recv_7(dt,ecu,m7);
send_8(ecu,dt,m8);

claim_c1(ecu,Alive);
claim_c2(ecu,Weakagree);
claim_c3(ecu,Niagree);
claim_c4(ecu,Nisynch);
claim_c5(ecu,SKR,Mk);
claim_c6(ecu,Secret,Na);
claim_c7(ecu,Secret,ssk);
claim_c8(ecu,Secret,smk);
claim_c9(ecu,Secret,sk(ecu));
}
}
```

B.2.2 EVITA+ Protocol Phase II

```
hashfunction h;
usertype Timestamp;

protocol evita1(dt,ccu,oem,ecu){
macro m9 = RequestSSK,ts,{RequestSSK,ts}sk(dt);
macro m10={SSK}pskECU,ts,{{SSK}pskECU,ts}sk(oem);
macro m11 ={FrmB}SSK,ts,{{FrmB}SSK,ts}sk(oem);
macro m12=Fdr,ts,h({Fdr,ts}Mk);
macro m13=ack,ts,h({ack,ts}Mk);
macro m14={SSK}pskECU,ts,h({{SSK}pskECU,ts}Mk);
macro m15=ack,ts,h({ack,ts}Mk);
macro m16= {FrmA}SSK,ts,h({{FrmA}SSK,ts}Mk);
macro m17=requesttransferexit,ts,h({requesttransferexit,ts}Mk)
    ;
macro m18 ={FrmB}SSK,ts,h({{FrmB}SSK,ts}Mk);
macro m19=requesttransferexit,ts,h({requesttransferexit,ts}Mk)
    ;
macro m20={FrmB}SSK,ts,h({{FrmB}SSK,ts}Mk);
macro m21=requesttransferexit,ts,h({requesttransferexit,ts}Mk)
    ;
```

```
macro m22=ack,ts,h({ack,ts}Mk);

role dt{
fresh ts: Timestamp;
const RequestSSK: Data;
var ack:Data;
const Fdr;
const FrmB;
const requesttransferexit:Data;
fresh Mk;
var SSK;
const pskECU;

send_9(dt,oem,m9);
recv_10(oem,dt,m10);
recv_11(oem,dt,m11);
send_12(dt,ecu,m12);
recv_13(ecu,dt,m13);
send_14(dt,ecu,m14);
recv_15(ecu,dt,m15);
send_18(dt,ccu,m18);
send_19(dt,ccu,m19);
recv_22(ecu,dt,m22);

claim_a1(dt,Alive);
claim_a2(dt,Weakagree);
claim_a3(dt,Nisynch);
claim_a4(dt,Niagree);
claim_a5(dt,SKR,Mk);
claim_a6(dt,Secret,SSK);
claim_a7(dt,Secret,FrmB);
claim_a8(dt,Secret,sk(dt));
}

role oem{
fresh ts:Timestamp;
var RequestSSK: Data;
```

```
fresh SSK;
const pskECU;
fresh FrmB;

recv_9(dt,oem,m9);
send_10(oem,dt,m10);
send_11(oem,dt,m11);

claim_x1(oem,Alive);
claim_x2(oem,Weakagree);
claim_x3(oem,Nisynch);
claim_x4(oem,Niagree);
claim_x5(oem,Secret,SSK);
claim_x6(oem,Secret,sk(oem));
}

role ccu{
var FrmA;
var FrmB;
var requesttransferexit:Data;
fresh ts:Timestamp;
var Mk;
var SSK;
const pskECU;

recv_16(ecu,ccu,m16);
recv_17(ecu,ccu,m17);
recv_18(dt,ccu,m18);
recv_19(dt,ccu,m19);
send_20(ccu,ecu,m20);
send_21(ccu,ecu,m21);

claim_b1(ccu,Alive);
claim_b2(ccu,Weakagree);
claim_b3(ccu,Nisynch);
claim_b4(ccu,Niagree);
claim_b5(ccu,Secret,FrmA);
```

```
claim_b6(ccu,Secret,FrmB);
claim_b7(ccu,SKR,Mk);
claim_b8(ccu,Secret,sk(ccu));
}

role ecu{
fresh ts:Timestamp;
const ack:Data;
const FrmA;
var FrmB;
var Fdr;
fresh requesttransferexit:Data;
var Mk;
var SSK;
const pskECU;

recv_12(dt,ecu,m12);
send_13(ecu,dt,m13);
recv_14(dt,ecu,m14);
send_15(ecu,dt,m15);
send_16(ecu,ccu,m16);
send_17(ecu,ccu,m17);
recv_20(ccu,ecu,m20);
recv_21(ccu,ecu,m21);
send_22(ecu,dt,m22);

claim_c1(ecu,Alive);
claim_c2(ecu,Weakagree);
claim_c3(ecu,Nisynch);
claim_c4(ecu,Niagree);
claim_c5(ecu,Secret,FrmA);
claim_c6(ecu,Secret,FrmB);
claim_c7(ecu,SKR,Mk);
claim_c8(ecu,Secret,sk(ecu));
}
}
```

B.3 Scyther Scripts for FOTA Protocol

These scripts are for the protocol described in section 5.6.

B.3.1 FOTA Protocol Phase I

```
hashfunction h;
usertype Timestamp;

protocol FOTAdownload(md,oem){
macro m1 = md,{KNa}pk(oem),ts,{md,{KNa}pk(oem),ts}sk(md);
macro m2 = oem,ack,ts,{oem,ack,ts}sk(oem);
macro m3 = md,RequestDL,idecu,ts,h({md,RequestDL,idecu,ts}KNa)
    ;
macro m4 = oem,{pskUNLOCK}pskECU,ts,h({oem,{pskUNLOCK}pskECU,
    ts}KNa);
macro m5 = md,ack,ts,h({md,ack,ts}KNa);
macro m6 = oem,{frmnew}pskFEK,ts,h({oem,{frmnew}pskFEK,ts}KNa)
    ;
macro m7 = md,ack,ts,h({md,ack,ts}KNa);

role md{
fresh idecu:Data;
fresh ts: Timestamp;
fresh RequestDL: Data;
var pskFEK;
var pskUNLOCK;
var pskECU;
var frmnew;
const ack;
fresh KNa;

send_1(md,oem,m1);
recv_2(oem,md,m2);
send_3(md,oem,m3);
recv_4(oem,md,m4);
send_5(md,oem,m5);
recv_6(oem,md,m6);
```

```
send_7(md,oem,m7);

claim_a1(md,Alive);
claim_a2(md,Weakagree);
claim_a3(md,Nisynch);
claim_a4(md,Niagree);
claim_a5(md,Secret,pskFEK);
claim_a6(md,Secret,frmnew);
claim_a7(md,SKR,KNa);
claim_a8(md,Secret,sk(md));
}

role oem{
var idecu:Data;
fresh ts:Timestamp;
var RequestDL: Data;
fresh frmnew;
const ack;
fresh pskFEK;
fresh pskECU;
fresh pskUNLOCK;
var KNa;

recv_1(md,oem,m1);
send_2(oem,md,m2);
recv_3(md,oem,m3);
send_4(oem,md,m4);
recv_5(md,oem,m5);
send_6(oem,md,m6);
recv_7(md,oem,m7);

claim_x1(oem,Alive);
claim_x2(oem,Weakagree);
claim_x3(oem,Nisynch);
claim_x4(oem,Niagree);
claim_x5(oem,Secret,pskUNLOCK);
claim_x6(oem,Secret,KNa);
```

```
claim_x7(oem,Secret,pskFEK);
claim_x8(oem,Secret,sk(oem));
}
}
```

B.3.2 FOTA Protocol Phase II

```
hashfunction h;
function f;
usertype Timestamp;

protocol FOTAinstall(md,ccu,ecu){
macro m8 = md,{KNb}pk(ccu),ts,{md,{KNb}pk(ccu),ts}sk(md);
macro m9 = ccu,{KNb}pk(ecu),ts,{ccu,{KNb}pk(ecu),ts}sk(ccu);
macro m10 = ecu,ack,ts,h({ecu,ack,ts}KNb);
macro m11 = ccu,ack,ts,h({ccu,ack,ts}KNb);
macro m12 = md,{pskUNLOCK}pskECU,ts,h({md,{pskUNLOCK}pskECU,ts
    }KNb);
macro m13 = ecu,ack,ts,h({ecu,ack,ts}KNb);
macro m14 = ecu,{frmold}pskFEK,ts,h({ecu,{frmold}pskFEK,ts}KNb
    );
macro m15 = md,ack,ts,h(md,ack,ts);
macro m16 = md,{frmnew}pskFEK,ts,h({md,{frmnew}pskFEK,ts}KNb);
macro m17 = ecu,ack,ts,h({ecu,ack,ts}KNb);

role md{
fresh KNb:Nonce;
var ack: Data;
fresh ts:Timestamp;
fresh pskECU;
fresh pskUNLOCK;
fresh pskFEK;
var frmold;
fresh frmnew;

send_8(md,ccu,m8);
recv_11(ccu,md,m11);
send_12(md,ecu,m12);
recv_13(ecu,md,m13);
recv_14(ecu,md,m14);
send_15(md,ecu,m15);
send_16(md,ecu,m16);
recv_17(ecu,md,m17);

claim_a1(md,Alive);
claim_a2(md,Weakagree);
claim_a3(md,Niagree);
claim_a4(md,Nisynch);
claim_a5(md,SKR,KNb);
claim_a6(md,Secret,pskECU);
claim_a7(md,Secret,pskFEK);
claim_a8(md,Secret,pskUNLOCK);
claim_a9(md,Secret,frmold);
claim_a10(md,Secret,frmnew);
claim_a11(md,Secret,sk(md));
}

role ccu{
var KNb:Nonce;
var ack: Data;
fresh ts:Timestamp;

recv_8(md,ccu,m8);
send_9(ccu,ecu,m9);
recv_10(ecu,ccu,m10);
send_11(ccu,md,m11);

claim_b1(ccu,Alive);
claim_b2(ccu,Weakagree);
claim_b3(ccu,Niagree);
claim_b4(ccu,Nisynch);
claim_b5(ccu,SKR,KNb);
claim_b6(ccu,Secret,sk(ccu));
}

```

```
role ecu{
var KNb:Nonce;
const ack: Data;
fresh ts:Timestamp;
fresh pskECU;
fresh pskUNLOCK;
fresh pskFEK;
fresh frmold;
var frmnew;

recv_9(ccu,ecu,m9);
send_10(ecu,ccu,m10);
recv_12(md,ecu,m12);
send_13(ecu,md,m13);
send_14(ecu,md,m14);
recv_15(md,ecu,m15);
recv_16(md,ecu,m16);
send_17(ecu,md,m17);

claim_c1(ecu,Alive);
claim_c2(ecu,Weakagree);
claim_c3(ecu,Niagree);
claim_c4(ecu,Nisynch);
claim_c5(ecu,Secret,pskECU);
claim_c6(ecu,Secret,pskFEK);
claim_c7(ecu,Secret,pskUNLOCK);
claim_c8(ecu,Secret,frmold);
claim_c9(ecu,Secret,frmnew);
claim_c10(ecu,SKR,KNb);
claim_c11(ecu,Secret,sk(ecu));
}
}
```

B.3.3 FOTA Protocol Replacement Phase

```
hashfunction h;
function f;
usertype Timestamp;
```

```

protocol FOTAupdate(md, ccu , ecu , oem) {
macro m1 = md, RequestID , ts , {md, RequestID , ts } sk (md) ;
macro m2 = ccu , RequestID , ts ;
macro m3 = ecu , idecu , ts ;
macro m4 = ccu , idecu , ts ;
macro m5 = md, RequestVerifyID , idecu , ts , {md, RequestVerifyID ,
    idecu , ts } sk (md) ;
macro m6 = oem, { rnd , pskECU, pk ( ecu ) }pk ( ccu ) , ts , { oem, { rnd , pskECU
    , pk ( ecu ) }pk ( ccu ) , ts } sk (oem) ;
macro m7 = md, { rnd , pskECU, pk ( ecu ) }pk ( ccu ) , ts , {md, { rnd , pskECU,
    pk ( ecu ) }pk ( ccu ) , ts } sk (md) ;
macro m8 = ccu , rnd , pk ( ccu ) , ts ;
macro m9 = ecu , { { rnd }pskECU}pk ( ccu ) , ts ;
macro m10 = ccu , ack , ts , { ccu , ack , ts } sk ( ccu ) ;

role md{
var ack : Data ;
fresh ts : Timestamp ;
fresh RequestID , RequestVerifyID : Data ;
var idecu : Data ;
var pskECU;
var rnd ;

send_1 (md, ccu ,m1) ;
recv_4 ( ccu ,md,m4) ;
send_5 (md, oem ,m5) ;
recv_6 ( oem ,md,m6) ;
send_7 (md, ccu ,m7) ;
recv_10 ( ccu ,md, m10) ;

claim_a1 (md, Alive ) ;
claim_a2 (md, Weakagree ) ;
claim_a3 (md, Niagree ) ;
claim_a4 (md, Nisynch ) ;
claim_a5 (md, Secret , sk (md) ) ;
}
```

```

role ccu{
fresh ack: Data;
fresh ts:Timestamp;
var RequestID:Data;
var idecu:Data;
var pskECU;
fresh pskCCU;
var rnd;

recv_1(md,ccu,m1);
send_2(ccu,ecu,m2);
recv_3(ecu,ccu,m3);

send_4(ccu,md,m4);
recv_7(md,ccu,m7);
send_8(ccu,ecu,m8);
recv_9(ecu,ccu,m9);
send_10(ccu,md,m10);

claim_b1(ccu,Alive);
claim_b2(ccu,Weakagree);
claim_b3(ccu,Niagree);
claim_b4(ccu,Nisynch);
claim_b5(ccu,Secret,pskECU);
claim_b6(ccu,Secret,sk(ccu));
}

role ecu{
fresh ack: Data;
fresh ts:Timestamp;
var RequestID:Data;
fresh idecu:Data;
fresh pskECU;
fresh pskCCU;
var rnd;

recv_2(ccu,ecu,m2);
send_3(ecu,ccu,m3);
recv_8(ccu,ecu,m8);
send_9(ecu,ccu,m9);

claim_c1(ecu,Alive);
claim_c2(ecu,Weakagree);
claim_c3(ecu,Niagree);
claim_c4(ecu,Nisynch);
claim_c5(ecu,Secret,pskECU);
claim_c6(ecu,Secret,sk(ecu));
}

role oem{
fresh ack: Data;
fresh ts:Timestamp;
var idecu:Data;
var RequestVerifyID:Data;
fresh pskECU;
fresh rnd;

recv_5(md,oem,m5);
send_6(oem,md,m6);

claim_d1(oem,Alive);
claim_d2(oem,Weakagree);
claim_d3(oem,Niagree);
claim_d4(oem,Nisynch);
claim_d5(oem,Secret,pskECU);
claim_d6(oem,Secret,sk(md));
}
}
```

B.4 Scyther Scripts for DiaLOG Protocol

These scripts are for the protocol described in section 6.6.

B.4.1 DiaLOG Full Authorisation Protocol

```
hashfunction h;
const fullreq;

protocol DiaLOGFull(mo,ccu){
macro m1 = mo,{ccu,fullreq,nmo}k(mo,ccu);
macro m2 = ccu,{mo,ks,nmo}k(mo,ccu);
macro m3 = mo,{ccu,reqdtc}ks;
macro m4 = {ccu,dtc}ks,{{ccu,dtc}ks}sk(ccu);

role mo{
fresh nmo:Nonce;
var ks:SessionKey;
var dtc:Data;
fresh reqdtc:Data;

send_1(mo,ccu,m1);
recv_2(ccu,mo,m2);
send_3(mo,ccu,m3);
recv_4(ccu,mo,m4);

claim_a1(mo,Alive);
claim_a2(mo,Weakagree);
claim_a3(mo,Niagree);
claim_a4(mo,Nisynch);
claim_a5(mo,SKR,nmo);
claim_a6(mo,Secret,ks);
claim_a7(mo,Secret,k(mo,ccu));
}

role ccu{
var nmo:Nonce;
fresh ks:SessionKey;
fresh dtc:Data;
var reqdtc:Data;

recv_1(mo,ccu,m1);
send_2(ccu,mo,m2);
recv_3(mo,ccu,m3);
send_4(ccu,mo,m4);

claim_b1(ccu,Alive);
claim_b2(ccu,Weakagree);
claim_b3(ccu,Niagree);
claim_b4(ccu,Nisynch);
claim_b5(ccu,SKR,nmo);
claim_b6(ccu,Secret,ks);
claim_b7(ccu,Secret,k(mo,ccu));
}
}
```

B.4.2 DiaLOG Basic Authorisation Protocol

```
hashfunction h;
const basicreq;

protocol basic1DiaLOG(mt,mo,ccu)
{
macro m1 = mt,mo,{ccu,basicreq,nmt}k(mo,mt);
macro m2 = mo,ccu,{mt,basicreq,nmo,nmt}k(mo,ccu);
macro m3 = ccu,mo,{mt,ks,nmo,nc,nmt}k(mo,ccu);
macro m4 = mo,mt,{mt,ccu,ks,nc,nmt,nmo}k(mo,mt);
macro m5 = mt,ccu,{mt,mo,ccu,nc,reqdtc}ks;
macro m6 = {ccu,dtc}ks,{{ccu,dtc}ks}sk(ccu);

role mt
{
fresh nmt: Nonce;
var nmo:Nonce;
var ks:SessionKey;
var nc:Nonce;
var dtc:Data;
fresh reqdtc:Data;

send_1(mt,mo,m1);
recv_4(mo,mt,m4);
send_5(mt,ccu,m5);
recv_6(ccu,mt,m6);
claim_R1(mt,Alive);
claim_R2(mt,Weakagree);
claim_R3(mt,Nisynch);
claim_R4(mt,Niagree);
claim_R5(mt,Secret, ks);
claim_R6(mt,Secret, k(mo,mt));
claim_R7(mt,SKR, nmo);
claim_R8(mt,SKR, nc);
claim_R9(mt,SKR, nmt);
}

role mo
{
fresh nmo:Nonce;
var nmt,nc:Nonce;
var ks:SessionKey;

recv_1(mt,mo,m1);
send_2(mo,ccu,m2);
recv_3(ccu,mo,m3);
send_4(mo,mt,m4);

claim_I1(mo,Alive);
claim_I2(mo,Weakagree);
claim_I3(mo,Nisynch);
claim_I4(mo,Niagree);
claim_I5(mo,Secret, ks);
claim_I6(mo,Secret, k(mo,mt));
claim_I7(mo,Secret, k(mo,ccu));
claim_I8(mo,SKR, nmo);
claim_I9(mo,SKR, nc);
claim_I10(mo,SKR, nmt);
}
```

```

role ccu
{
var nmo,nmt: Nonce;
fresh nc:Nonce;
fresh ks:SessionKey;
fresh dtc:Data;
var reqdtc:Data;

recv_2(mo,ccu,m2);
send_3(ccu,mo,m3);
recv_5(mt,ccu,m5);
send_6(ccu,mt,m6);

claim_C1(ccu,Alive);
claim_C2(ccu,Weakagree);
claim_C3(ccu,Nisynch);
claim_C4(ccu,Niagree);
claim_C5(ccu,Secret, ks);
claim_C6(ccu,Secret, k(mo,ccu));
claim_C7(ccu,SKR, nmo);
claim_C8(ccu,SKR, nc);
claim_C9(ccu,SKR, nmt);
}
}
```

B.5 Scyther Scripts for AutoLOG Protocol

These scripts are for the protocol described in section 7.6.

B.5.1 AutoLOG Protocol Phase I

```
usertype SessionKey;
const Fresh: Function;
hashfunction MAC;
function f;
usertype Timestamp;

```

```
protocol AutoLOG1(dt,ccu){
macro m1 = dt,{{ccu,serviceupdate,na}sk(dt)}pk(ccu);
macro m2 = ccu,{{dt,ack,na,nb}sk(ccu)}pk(dt);
macro m3 = dt,ccu,servicetype,servicedate,nextdate,ts,MAC(f(na
    ,nb),ccu,servicetype,servicedate,nextdate,ts);
macro m4 = ccu,dt,ack,MAC(f(na,nb),dt,ack);
macro m5 = dt,s1,s2,s3,MAC(f(na,nb),s1,s2,s3);

role dt
{
fresh na: Nonce;
var nb: Nonce ;
fresh servicetype,servicedate,nextdate:Data;
var ack:Data;
fresh serviceupdate:Data;
fresh ts:Timestamp;
fresh s1,s2,s3:Data;

send_1(dt,ccu,m1);
recv_2(ccu,dt,m2);
send_3(dt,ccu,m3);
recv_4(ccu,dt,m4);
send_5(dt,ccu,m5);

claim_I1(dt,Alive);
claim_I2(dt,Weakagree);
claim_I3(dt,Nisynch);
claim_I4(dt,Niagree);
claim_I5(dt,SKR, na);
}

role ccu
{
var na: Nonce;
fresh nb: Nonce;
var servicetype,servicedate,nextdate:Data;
fresh ack:Data;
var serviceupdate:Data;
var ts:Timestamp;
var s1,s2,s3:Data;

recv_1(dt,ccu,m1);
send_2(ccu,dt,m2);
recv_3(dt,ccu,m3);
send_4(ccu,dt,m4);
recv_5(dt,ccu,m5);

claim_R4(ccu,Alive);
claim_R6(ccu,Weakagree);
claim_R1(ccu,Nisynch);
claim_R2(ccu,Niagree);
claim_R3(ccu,SKR, nb);
}
}
```

B.5.2 AutoLOG Protocol Phase II

```
hashfunction MAC;
function f;

protocol AutoLOG2(ccu,ecu){
macro m6 = ccu,{ecu,validateservice,nc}k(ccu,ecu);
macro m7 = ecu,{ccu,ackready,nc,nd}k(ccu,ecu);
macro m8 = ccu,ecu,s1,s2,s3,MAC(f(nc,nd),ecu,s1,s2,s3);
macro m9 = ecu,ccu,acke,MAC(f(nc,nd),ccu,acke);

role ccu
{
fresh s1,s2,s3:Data;
var acke,ackready:Data;
fresh nc: Nonce;
var nd:Nonce;
fresh validateservice:Data;

send_6(ccu,ecu,m6);
recv_7(ecu,ccu,m7);
send_8(ccu,ecu,m8);
recv_9(ecu,ccu,m9);

claim_R1(ccu,Alive);
claim_R2(ccu,Weakagree);
claim_R3(ccu,Nisynch);
claim_R4(ccu,Niagree);
claim_R5(ccu,SKR,nc);
claim_R6(ccu,Secret,k(ccu,ecu));
}

role ecu
{
var s1,s2,s3:Data;
fresh ackready,acke:Data;
var nc: Nonce;
fresh nd:Nonce;
var validateservice:Data;

recv_6(ccu,ecu,m6);
send_7(ecu,ccu,m7);
recv_8(ccu,ecu,m8);
send_9(ecu,ccu,m9);

claim_E1(ecu,Alive);
claim_E2(ecu,Weakagree);
claim_E3(ecu,Nisynch);
claim_E4(ecu,Niagree);
claim_E5(ecu,SKR,nd);
claim_E6(ecu,Secret,k(ccu,ecu));
}
}
```

B.5.3 AutoLOG Protocol Phase III

```
usertype SessionKey;
const Fresh: Function;
```

```
hashfunction MAC;
function f;
usertype Timestamp;

protocol AutoLOG3(md,ccu){
macro m10 = md,{serviceupdatereq,ne}k(md,ccu);
macro m11 = ccu,{md,serviceupdate,nf}k(md,ccu);
macro m12 = md,ccu,{ack}f(ne,nf);
macro m13 = {ccu,md,servicetype,servicedate,nextdate,ts}f(ne,
    nf),{{ccu,md,servicetype,servicedate,nextdate,ts}f(ne,nf)}
    sk(ccu);
macro m14 = {md,ccu,ackm}f(ne,nf);
macro m15 = ccu,{s1,s2,s3}f(ne,nf),{{s1,s2,s3}f(ne,nf)}sk(ccu)
    ;
macro m16 = {md,ackm}f(ne,nf);

role md
{
fresh ne: Nonce;
var nf: Nonce ;
var servicetype,servicedate,nextdate:Data;
fresh ackm,ack:Data;
var serviceupdate:Data;
var ts:Timestamp;
fresh serviceupdatereq:Data;
var s1,s2,s3:Data;

send_10(md,ccu,m10);
recv_11(ccu,md,m11);
send_12(md,ccu,m12);
recv_13(ccu,md,m13);
send_14(md,ccu,m14);
recv_15(ccu,md,m15);
send_16(md,ccu,m16);

claim_I1(md,Alive);
claim_I2(md,Nisynch);
claim_I3(md,Niagree);
claim_I4(md,Weakagree);
claim_I5(md,SKR, ne);
claim_I6(md,Secret, k(md,ccu));
}

role ccu
{
var ne: Nonce;
fresh nf: Nonce;
fresh servicetype,servicedate,nextdate:Data;
var ack,ackm:Data;
fresh serviceupdate:Data;
fresh ts:Timestamp;
var serviceupdatereq:Data;
fresh s1,s2,s3:Data;

recv_10(md,ccu,m10);
send_11(ccu,md,m11);
recv_12(md,ccu,m12);
send_13(ccu,md,m13);
recv_14(md,ccu,m14);
send_15(ccu,md,m15);
recv_16(md,ccu,m16);

claim_R1(ccu,Alive);
claim_R2(ccu,Weakagree);
claim_R3(ccu,Nisynch);
claim_R4(ccu,Niagree);
claim_R5(ccu,SKR, nf);
claim_R6(ccu,Secret, k(md,ccu));
}
}
```

Bibliography

[1] E. Uhlemann, "Introducing connected vehicles," *IEEE Vehicular Technology Magazine*, vol. 10, pp. 23–31, Mar. 2015.

[2] A. Paul, R. Chauhan, R. Srivastava, and M. Baruah, "Advanced Driver Assistance Systems," tech. rep., SAE Technical Paper, Feb. 2016. `http://papers.sae.org/2016-28-0223/`,"Last visited on 20/09/2016".

[3] E. Commission, "eCall: Time saved = lives saved." `https://ec.europa.eu/digital-single-market/ecall-time-saved-lives-saved`, Jan. 2016. Last visited on 20/09/2016.

[4] E. Commission, "eCall in all new cars from April 2018." https://ec.europa.eu/digital-single-market/en/news/ecall-all-new-cars-april-2018, Apr. 2015. Last visited on 19/09/2016.

[5] I. C. Society, "IEEE 802.11p." https://www.ietf.org/mail-archive/web/its/current/pdfqf992dHy9x.pdf, July 2010.

[6] Informa-PLC, "Connected car USA." `https://usa.connectedcarsworld.com/`. Last visited on 19/09/2016.

[7] S. Norton, "How Ford is building the connected car." `http://www.wsj.com/articles/how-ford-is-building-the-connected-car-1456110337`, Feb. 2016. Last visited on 19/09/2016.

[8] Nissan, "Put down the phone and stay connected." `http://www.nissanusa.com/connect`. Last visited on 19/09/2016.

[9] K. Fitchard, "A peek into GMs connected car future." `http://fortune.com/2015/03/27/a-peek-into-gms-connected-car-future/`, Mar. 2015. Last visited on 19/09/2016.

[10] J.-P. Hubaux, S. Capkun, and J. Luo, "The security and privacy of smart vehicles," *IEEE Security & Privacy Magazine*, vol. 2, no. LCA-ARTICLE-2004-007, pp. 49–55, 2004.

[11] C. Miller and C. Valasek, "Adventures in automotive networks and control units," in *DEF CON 21 Hacking Conference. Las Vegas, NV: DEF CON*, (Las Vegas, Nevada, USA), pp. 260–264, DEF CON, Aug 1-4 2013.

[12] S. Checkoway, D. McCoy, B. Kantor, D. Anderson, H. Shacham, S. Savage, K. Koscher, A. Czeskis, F. Roesner, and T. Kohno, "Comprehensive experimental analyses of automotive attack surfaces," in *USENIX Security Symposium*, (San Francisco, California, USA), USENIX Association, Aug 8-12 2011. howpublished=http://static.usenix.org/events/sec11/tech/full_papers/Checkoway.pdf, Last visited on 19/09/2016.

[13] F. Sagstetter, M. Lukasiewycz, 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 Conference on Design, Automation and Test in Europe*, (San Jose, CA, USA), pp. 458–463, ACM, 2013.

[14] S. Chockalingam and H. S. Lallie, "Alarming! Security aspects of the wireless vehicle: Review," *International Journal of Cyber-Security and Digital Forensics (IJCSDF)*, vol. 3, pp. 200–208, Aug. 2014.

[15] G. Andy, "Hackers remotely kill a jeep on a highway - with me in it." https://www.wired.com/2015/07/hackers-remotely-kill-jeep-highway/, July 2015. Last visited on 19/09/2016.

[16] 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 *Security and Privacy (SP), 2010 IEEE Symposium on*, (Berkeley/Oakland, California, USA), pp. 447–462, IEEE, May 16-19 2010.

[17] 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 *19th USENIX Security Symposium*, (Washington DC), pp. 323–328, USENIX Association, August 11-13 2010. https://www.usenix.org/legacy/event/sec10/tech/full_papers/Rouf.pdf.

[18] I. Studnia, V. Nicomette, E. Alata, Y. Deswarte, M. Kaâniche, and Y. Laarouchi, “Survey on security threats and protection mechanisms in embedded automotive networks,” in *Dependable Systems and Networks Workshop (DSN-W), 2013 43rd Annual IEEE/IFIP Conference on*, pp. 1–12, IEEE, 2013.

[19] N. Lu, N. Cheng, N. Zhang, X. Shen, and J. W. Mark, “Connected vehicles: Solutions and challenges,” *IEEE internet of things journal*, vol. 1, no. 4, pp. 289–299, 2014.

[20] K. Lemke, C. Paar, and M. Wolf, *Embedded security in cars*. Bochum, Germany: Springer Berlin Heidelberg, Oct. 2006.

[21] O. Henniger, “EVITA : E-Safety Vehicle Intrusion Protected Applications,” techreport, EVITA, Brussels, Apr. 2011. http://www.evita-project.org/EVITA_factsheet.pdf, Last visited on 19/09/2016.

[22] J. P. Stotz, N. Bimeyer, F. Kargl, S. Dietze, P. Papadimitratos, and C. Schleiffer, “Security requirements of vehicle security architecture,” Tech. Rep. Deliverable D1.1, PRESERVE, June 2011. http://tinyurl.com/ht6j54b, Last visited on 19/09/2016.

[23] A. Kung, “Security architecture and mechanisms for V2V/V2I,” Tech. Rep. Deliverable 2.1, SeVeCom, Aug. 2008. Last visited on 19/09/2016.

[24] D. Geronimo, A. M. Lopez, A. D. Sappa, and T. Graf, “Survey of pedestrian detection for advanced driver assistance systems,” *IEEE transactions on pattern analysis and machine intelligence*, vol. 32, pp. 1239–1258, July 2010.

[25] M. Wolf, A. Weimerskirch, and T. Wollinger, “State of the art: Embedding security in vehicles,” *EURASIP Journal on Embedded Systems*, vol. 2007, pp. 1–16, Dec. 2007.

[26] E. Uhlemann, “Autonomous vehicles are connecting,” *IEEE Vehicular Technology Magazine*, vol. 10, pp. 22–25, June 2015.

[27] J. Schäuffele and T. Zurawka, *Automotive software engineering: Principles, processes, methods and tools*. Warrendale, Pa. : Society of Automotive Engineers, 1st ed., June 2005.

[28] B. Fleming, “An overview of advances in automotive electronics,” *Vehicular Technology Magazine, IEEE*, vol. 9, pp. 4–19, Mar. 2014.

[29] AUTOSAR, "AUTOSAR." http://www.autosar.org/about/technical-overview/, 2014. Last visited on 19/09/2016.

[30] M. Wolf, *Security Engineering for Vehicular IT Systems*. Germany: SpringerLink, 1st ed., 2009.

[31] M. S. Idrees, H. Schweppe, Y. Roudier, M. Wolf, D. Scheuermann, and O. Henniger, "Secure automotive on-board protocols: A case of over-the-air firmware updates," in *Communication Technologies for Vehicles* (T. Strang, A. Festag, A. Vinel, R. Mehmood, C. R. Garcia, and M. Rckl, eds.), vol. 6596 of *LNCS*, pp. 224–238, Oberpfaffenhofen, Germany: Springer Berlin Heidelberg, March 23-24 2011.

[32] TCG, "TCG TPM 2.0 Automotive thin profile," Specification 1.0, Trusted Computing Group, Mar. 2015. https://www.trustedcomputinggroup.org/wp-content/uploads/TCG-TPM-2.0-Automotive-Thin-Profile_v1.0.pdf, Last visited on 19/09/2016.

[33] Fujitsu, "Secure Hardware Extension: Data security for automotive embedded systems." https://www.escrypt.com/fileadmin/escrypt/pdf/WEB_Secure_Hardware_Extension_Wiewesiek.pdf, Feb. 2012. Last visited on 19/09/2016.

[34] E. Barker and J. Kelsey, "Recommendation for Random Number Generation using deterministic random bit generators." http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-90Ar1.pdf, June 2015.

[35] "Recommended elliptic curves for federal government use." http://csrc.nist.gov/groups/ST/toolkit/documents/dss/NISTReCur.pdf, July 1999.

[36] RSALaboratories, "PKCS #1 v2.2: RSA cryptography standard." https://www.emc.com/collateral/white-papers/h11300-pkcs-1v2-2-rsa-cryptography-standard-wp.pdf, 2012.

[37] "Specification for the Advanced Encryption Standard (FIPS 197)." http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.197.pdf, Nov. 2001.

[38] M. Dworkin, "Recommendation for block cipher modes of operation:the CMAC mode for authentication." http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-38B.pdf, May 2005.

[39] C. M. Gutierrez and J. M. Turner, "The keyed-hash message authentication code (HMAC)." http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.198-1.pdf, July 2008.

[40] P. Pritzker and W. E. May, "Secure hash standard (SHS)." http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.180-4.pdf, Aug. 2015.

[41] "ISO/IEC 10118-3:2004 information technology – security techniques – hash-functions – part 3: Dedicated hash-functions," Mar. 2004.

[42] Infineon, "TC1798 32-bit single-chip microcontroller." http://tinyurl.com/h6qqwl3, May 2014. Last visited on 19/09/2016.

[43] NXP, "MPC5646C microcontroller data sheet." http://tinyurl.com/zxen327, Feb. 2014. Last visited on 19/09/2016.

[44] M. Wolf and T. Gendrullis, "Design, implementation, and evaluation of a vehicular hardware security module," in *Information Security and Cryptology-ICISC 2011* (H. Kim, ed.), vol. 7259 of *LNCS*, pp. 302–318, Seoul, Korea: Springer, Nov 30 - Dec 2 2012.

[45] Infineon, "AURIX family tc297ta." http://tinyurl.com/zkxb2rl, 2015. Last visited on 19/09/2016.

[46] NXP, "MPC5748G microcontroller data sheet." http://tinyurl.com/hnbd9z8, May 2015. Last visited on 19/09/2016.

[47] TCG, "Trusted computing group and automotive security." https://www.trustedcomputinggroup.org/wp-content/uploads/TCG-Automotive-Security-FAQ.pdf. Last visited on 16/11/2016.

[48] S. Kotani and I. McDonald, "Automotive TPM." http://www.trustedcomputinggroup.org/securing-connected-auto/, Aug. 2015. Last visited on 19/09/2016.

[49] M. Ruff, "Evolution of Local Interconnect Network (LIN) solutions," in *Vehicular Technology Conference, 2003. VTC 2003-Fall. 2003 IEEE 58th*, vol. 5, pp. 3382–3389, IEEE, IEEE, Oct 6-9 2003.

[50] M. Cooperation, "Media Oriented Systems Transport specifications," Specification Rev 2.4, MOST Cooperation, May 2005.

[51] R. Makowitz and C. Temple, "FlexRay- A communication network for automotive control systems," in *2006 IEEE International Workshop on Factory Communication Systems*, pp. 207–212, Freescale Semiconductor, 2006.

[52] M. Wolf, A. Weimerskirch, and C. Paar, "Security in automotive bus systems," in *Workshop on Embedded Security in Cars*, Nov. 2004.

[53] ISO, "Road vehicles – controller area network (CAN) – part 1: Data link layer and physical signalling," Feb. 2013.

[54] C. Miller and C. Valasek, "A survey of remote automotive attack surfaces," *Black Hat USA. Las Vegas, NV*, 2014.

[55] C. Miller and C. Valasek, "Remote exploitation of an unaltered passenger vehicle," *Black Hat USA. Las Vegas, NV*, Aug. 2015.

[56] I. Foster and K. Koscher, "Exploring controller area networks," *USENIX*, vol. 40, pp. 6–11, Dec. 2015. https://www.usenix.org/system/files/login/articles/login_dec15_02_foster.pdf, Last visited on 19/09/2016.

[57] T. Zhang, H. Antunes, and S. Aggarwal, "Defending connected vehicles against malware: Challenges and a solution framework," *Internet of Things Journal, IEEE*, vol. 1, pp. 10–21, Feb. 2014.

[58] C. Solomon and B. Groza, "LiMon - Lightweight authentication for tire pressure monitoring sensors," in *1st Workshop on the Security of Cyber-Physical Systems (affiliated to ESORICS 2015)*, LNCS, (Vienna, Austria), Springer, Sept. 2015.

[59] A. I. Alrabady and S. M. Mahmud, "Analysis of attacks against the security of keyless-entry systems for vehicles and suggestions for improved designs," *Vehicular Technology, IEEE Transactions on*, vol. 54, pp. 41–50, Jan. 2005.

[60] S. Indesteege, N. Keller, O. Dunkelman, E. Biham, and B. Preneel, "A practical attack on Keeloq," in *Advances in Cryptology–EUROCRYPT 2008* (N. Smart, ed.), vol. 4965 of *LNCS*, pp. 1–18, Istanbul,Turkey: Springer Berlin Heidelberg, Apr 13-17 2008.

[61] M. Kasper, T. Kasper, A. Moradi, and C. Paar, "Breaking Keeloq in a flash: On extracting keys at lightning speed," in *Progress in Cryptology–AFRICACRYPT 2009* (B. Preneel, ed.), vol. 5580 of *LNCS*, pp. 403–420, Gammarth, Tunisia: Springer, Jun 21-25 2009.

[62] A. Francillon, B. Danev, and S. Capkun, “Relay attacks on passive keyless entry and start systems in modern cars,” in *NDSS*, 2011.

[63] R. Verdult, F. D. Garcia, and J. Balasch, “Gone in 360 seconds: Hijacking with Hitag2,” in *21st USENIX Security Symposium (USENIX Security 12)*, pp. 237–252, USENIX Association, 2012.

[64] R. Verdult, F. D. Garcia, and B. Ege, “Dismantling megamos crypto: Wirelessly lockpicking a vehicle immobilizer,” in *Supplement to the 22nd USENIX Security Symposium (USENIX Security 13)*, (Washington, D.C.), pp. 703–718, USENIX Association, 2015.

[65] T. Hoppe, S. Kiltz, and J. Dittmann, “Security threats to automotive CAN networks - practical examples and selected short-term countermeasures,” in *Computer Safety, Reliability, and Security, 27th International Conference, SAFECOMP 2008* (M. D. Harrison and M.-A. Sujan, eds.), vol. 96 of *LNCS*, (Newcastle upon Tyne, UK), pp. 11–25, Springer Berlin Heidelberg, Sept 22-25 2011.

[66] S. Singh, “Cyber warfare in cars.” http://tinyurl.com/o9tcwa6, Oct. 2014. Last visited on 19/09/2016.

[67] Wired, “Hacker disables more than 100 cars remotely.” https://www.wired.com/2010/03/hacker-bricks-cars/, Mar. 2010. Last visited on 19/09/2016.

[68] L. J. Hoffman, K. Lawson-Jenkins, and J. Blum, “Trust beyond security: An expanded trust model,” *Communications of the ACM*, vol. 49, pp. 94–101, July 2006.

[69] S. Diewald, T. Leinmüller, B. Atanassow, L.-P. Breyer, and M. Kranz, “Mobile device integration with V2X communication,” in *Proceedings of the 19th World Congress on Intelligent Transport Systems (ITS)*, (Vienna, Austria), Oct 22-26 2012.

[70] S. V. Electrical and E. D. S. S. Committee, “SAE J1962 revised apr2002,” Standard SAEJ1962, SAE Vehicle Electrical and Electronics Diagnostics Systems Standards Committee, Apr. 2002.

[71] E. Electronics, “ELM327L.” http://www.elmelectronics.com/DSheets/ELM327L_Data_Sheet.pdf/. Last visited on 19/09/2016.

[72] U. I. Forum, “Universal serial bus cables and connectors class document,” techreport, Aug. 2007.

[73] "Mobile high-definition link (mhl)," techreport, MHLTech, Oct. 2013.

[74] "High-definition multimedia interface," Tech. Rep. 1.3, HDMI Licensing, LLC, Nov. 2006.

[75] M. La Polla, F. Martinelli, and D. Sgandurra, "A survey on security for mobile devices," *Communications Surveys & Tutorials, IEEE*, vol. 15, no. 1, pp. 446–471, 2013.

[76] T. Leinmüller, B. Atanassow, S. Diewald, L.-P. Breyer, and M. Kranz, "Security evaluation of mobile device integration with v2x communication," in *20th ITS World Congress*, (Tokyo, Japan), Citeseer, October 14-18 2013.

[77] A. Bouard, J. Schanda, D. Herrscher, and C. Eckert, "Automotive proxy-based security architecture for CE device integration," in *Mobile Wireless Middleware, Operating Systems, and Applications* (C. Borcea, P. Bellavista, C. Giannelli, T. Magedanz, and F. Schreiner, eds.), vol. 65 of *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*, pp. 62–76, Berlin, Germany: Springer, Nov 13-14 2012.

[78] N. Leavitt, "Mobile phones: The next frontier for hackers?," *Computer*, vol. 38, pp. 20–23, Apr. 2005.

[79] ARM, "ARM TrustZone." `https://www.arm.com/products/security-on-arm/trustzone`, 1995-2016.

[80] Intel, "Intel Software Guard Extensions." `https://software.intel.com/en-us/sgx`. Last visited on 19/09/2016.

[81] T. Topics, "Volvo trucks start european platoon demonstration tour." `http://www.ttnews.com/articles/volvo-trucks-start-european-platoon-demonstration-tour`, Mar. 2016.

[82] A. Cui, M. Costello, and S. J. Stolfo, "When firmware modifications attack: A case study of embedded exploitation," in *NDSS Symposium*, (San Diego, CA United States), Feb 24-27 2013.

[83] D. Barrera and P. Van Oorschot, "Secure software installation on smartphones," *IEEE Security & Privacy*, vol. 9, pp. 42–48, Nov. 2011.

[84] C. Paar and A. Weimerskirch, "Embedded security in a pervasive world," *information security technical report*, vol. 12, pp. 155–161, May 2007.

[85] J. S. Wey, J. Lüken, and J. Heiles, "Standardization activities for IPTV set-top box remote management," *IEEE Internet Computing*, vol. 13, pp. 32–39, May-Jun 2009.

[86] J. Hendricks and L. Van Doorn, "Secure bootstrap is not enough: Shoring up the trusted computing base," in *Proceedings of the 11th workshop on ACM SIGOPS European workshop*, (New York, USA), p. 11, ACM, 2004.

[87] K. Breitfelder and D. Messina, "IEEE 100 The authoritative dictionary of IEEE standards terms seventh edition," *IEEE Std 100-2000*, vol. 879, 2000.

[88] H. Software, "HIS flashloader specification," Tech. Rep. version 1.1, Herstellerinitiative Software, June 2006.

[89] Atmel, "32-bit AVR microcontroller automotive summary." http://www.atmel.com/images/doc9166s.pdf, Jan. 2012. Last visited on 19/09/2016.

[90] Renesas, "Renesas automotive." https://www.renesas.com/en-us/products/microcontrollers-microprocessors/rh850.html, 2012-2016. Last visited on 19/09/2016.

[91] D. Newcomb, "Tesla update strategy leaves other automakers in the dust." http://uk.pcmag.com/cars-products/40743/opinion/tesla-update-strategy-leaves-other-automakers-in-the-dust, Mar. 2015. Last visited on 20/09/2016.

[92] S. Hanley, "Musk responds to reveal of Tesla P100D, Hacking is a gift." http://www.teslarati.com/musk-responds-reveal-tesla-p100d-hacking/, Mar. 2016. Last visited on 20/09/2016.

[93] S. Duri, M. Gruteser, X. Liu, P. Moskowitz, R. Perez, M. Singh, and J.-M. Tang, "Framework for security and privacy in automotive telematics," in *Proceedings of the 2nd international workshop on Mobile commerce*, (New York, USA), pp. 25–32, ACM, 2002.

[94] S. Duri, J. Elliott, M. Gruteser, X. Liu, P. Moskowitz, R. Perez, M. Singh, and J.-M. Tang, "Data protection and data sharing in telematics," *Mobile networks and applications*, vol. 9, pp. 693–701, Dec. 2004.

[95] R. Schmidt-Cotta, H. Steffan, A. Kast, S. Labbett, and M. Brenner, "VERONICA: Vehicle event recording based on intelligent crash assessment," techreport, European Commission, Nov. 2006.

[96] R. Schmidt-Cotta, "Vehicle event recording based on intelligent crash assessment, VERONICA–ii final report," *European Commission Directorate-General for Energy and Transport, EC Contract No. TREN-07-ST-S07*, vol. 70764, p. 40, Oct. 2009.

[97] B. Canis and D. R. Peterman, "Black boxes in passenger vehicles: Policy issues," techreport, Congressional Research Service, July 2014.

[98] Y. Zhao, "Telematics: safe and fun driving," *IEEE Intelligent Systems*, vol. 17, no. 1, pp. 10–14, 2002.

[99] E. Commission, "eCall Do you have any concerns for your privacy? You shouldn't.." https://ec.europa.eu/digital-single-market/en/news/ecall-June 2014. Last visited on 19/09/2016.

[100] K. Chae, D. Kim, S. Jung, J. Choi, and S. Jung, "Evidence collecting system from car black boxes," in *Consumer Communications and Networking Conference (CCNC), 2010 7th IEEE*, (Las Vegas, Nevada, USA), pp. 1–2, IEEE, Jan 9-12 2010.

[101] D. Hynd and M. McCarthy, "Final report: Study on benefits resulting from the installation of Event Data Recorders." http://ec.europa.eu/transport/road_safety/pdf/vehicles/study_edr_2014.pdf, 2014. Last visited on 19/09/2016.

[102] K. Ryan, "Coming to a car dealership near you: Standardizing event data recorder technology use in automobiles," Technical report 90(3), IIT Chicago-Kent College of Law, June 2015.

[103] FIA, "My car my data." http://www.mycarmydata.eu/, Nov. 2015. Last visited on 19/09/2016.

[104] M. Cheminod, I. C. Bertolotti, L. Durante, R. Sisto, and A. Valenzano, "Experimental comparison of automatic tools for the formal analysis of cryptographic protocols," in *Dependability of Computer Systems, 2007. DepCoS-RELCOMEX '07. 2nd International Conference on* (W. Zamojski, J. Mazurkiewicz, J. Sugier, and T. Walkowiak, eds.), (Szklarska, Poland), IEEE, June 2007.

[105] C. J. F. Cremers, P. Lafourcade, and P. Nadeau, *Comparing State Spaces in Automatic Security Protocol Analysis*, pp. 70–94. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009.

[106] N. Dalal, J. Shah, K. Hisaria, and D. Jinwala, "A comparative analysis of tools for verification of security protocols," *International Journal of Communications, Network and System Sciences*, vol. 3, no. 10, p. 779, 2010.

[107] D. K. Nilsson, L. Sun, and T. Nakajima, "A framework for self-verification of firmware updates over the air in vehicle ECUs," in *GLOBECOM Workshops, 2008 IEEE*, (New Orleans, Louisiana), pp. 1–5, IEEE, IEEE, Nov 30 - Dec 4 2008.

[108] A. Adelsbach, U. Huber, and A.-R. Sadeghi, "Secure software delivery and installation in embedded systems," in *Embedded Security in Cars* (K. Lemke, C. Paar, and M. Wolf, eds.), ch. Embedded Security in Cars, pp. 27–49, Springer Berlin Heidelberg, 2006.

[109] A. Adelsbach, U. Huber, A.-R. Sadeghi, and C. Stüble, "Embedding trust into carssecure software delivery and installation," in *Third Workshop on Embedded Security in Cars (escar 2005), Cologne, Germany*, Oct 17 2005.

[110] C. Patsakis, K. Dellios, and M. Bouroche, "Towards a distributed secure in-vehicle communication architecture for modern vehicles," *Computers & Security*, vol. 40, pp. 60–74, Feb. 2014.

[111] "IEEE standard for test access port and boundary-scan architecture," May 2013.

[112] G. Pedroza, M. S. Idrees, L. Apvrille, and Y. Roudier, "A formal methodology applied to secure over-the-air automotive applications," in *Vehicular Technology Conference (VTC Fall), 2011 IEEE*, (San Francisco, California, USA), pp. 1–5, IEEE, Sept 5-8 2011.

[113] H. Software, "HIS functional specification of a flash driver," specification, Herstellerinitiative Software, June 2002.

[114] G. Lowe, "Casper: A compiler for the analysis of security protocols," *Journal of computer security*, vol. 6, no. 1, pp. 53–84, 1998.

[115] C. Cremers, "The Scyther tool: Verification, falsification, and analysis of security protocols," in *Computer Aided Verification* (A. Gupta and S. Malik, eds.), vol. 5123 of *LNCS*, (Princeton, NJ, USA), pp. 414–418, Springer Berlin Heidelberg, July 7-14 2008.

[116] G. Lowe, "A hierarchy of authentication specifications," in *Computer Security Foundations Workshop, 1997. Proceedings., 10th* (B. Werner, ed.), (Rockport, Massachusetts), pp. 31–43, IEEE, June 10-12 1997.

[117] C. Cremers and S. Mauw, *Operational semantics and verification of security protocols*. Springer Science & Business Media, illustrated ed., 2012.

[118] C. Cremers, *Scyther User Manual*, draft ed., Feb. 2014.

[119] C. J. Cremers, S. Mauw, and E. P. de Vink, "Injective synchronisation: an extension of the authentication hierarchy," *Theoretical Computer Science*, vol. 367, pp. 139–161, Nov. 2006.

[120] Microchip, "PIC32MZ embedded connectivity (EC) family." http://ww1.microchip.com/downloads/en/DeviceDoc/60001191F.pdf, July 2015. Last visited on 22/10/2016.

[121] Microchip, "CAN bus analyzer users guide." http://ww1.microchip.com/downloads/en/DeviceDoc/51848B.pdf, Aug. 2011. Last visited on 22/10/2016.

[122] Microchip, "MPLAB X integrated development environment (IDE) v3.20." http://www.microchip.com/mplab/mplab-x-ide, 1998-2016. Last visited on 19/09/2016.

[123] Microchip, "MPLAB harmony integrated software framework." http://www.microchip.com/mplab/mplab-harmony, 2015. Last visited on 19/09/2016.

[124] LeCroy, "LeCroy Waverunner 6000 series oscilloscopes: Operator's manual." http://cdn.teledynelecroy.com/files/manuals/wr6a-om-e_rev_g.pdf, Feb. 2007. Last visited on 19/09/2016.

[125] Microchip, "PIC32MZ embedded connectivity (EC) starter kit users guide." http://ww1.microchip.com/downloads/en/DeviceDoc/70005147A.pdf, Oct. 2013. Last visited on 22/10/2016.

[126] Microchip, "CAN/LIN/J2602 PICtail (plus) daughter board users guide." http://ww1.microchip.com/downloads/en/DeviceDoc/70319B.pdf, 2011. Last visited on 22/10/2016.

[127] Microchip, "PIC32MZ embedded connectivity (EC) adapter board information sheet." http://ww1.microchip.com/downloads/en/DeviceDoc/50002199A.pdf, Aug. 2013. Last visited on 22/10/2016.

[128] Microchip, "Starter kit I/O expansion board information sheet." http://ww1.microchip.com/downloads/en/DeviceDoc/51950A.pdf, Aug. 2010. Last visited on 22/10/2016.

[129] R. B. Software, "Updating car ECUs over-the-air (FOTA)," white paper, Red Bend Software, 2011. Last visited on 20/09/2016.

[130] F. Hartwich, "CAN with flexible data rate." http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.365.1901&rep=rep1&type=pdf, Mar. 2012.

[131] A. Smith, "Toyota recalls 2.1 million vehicles." http://money.cnn.com/2014/02/12/autos/toyota-prius-recall/, Feb. 2014. Last visited on 19/09/2016.

[132] Reuters, "Volvo recalls 59,000 cars over software fault." http://www.bbc.co.uk/news/world-europe-35622753, Feb. 2016. Last visited on 19/09/2016.

[133] BBC, "Samsung hit by latest smart TV issue." http://www.bbc.co.uk/news/technology-31642195, Feb. 2015. Last visited on 19/09/2016.

[134] Samsung, *How To Update Your TVs Firmware Via USB.* Samsung, May 2012.

[135] Samsung, *E-Manual.* Samsung, Dec. 2011.

[136] T. Register, "Game of moans: Sky coughs to BORKED set top box BALLS-UP." http://www.theregister.co.uk/2015/02/17/sky_set_top_box_broken/, Feb. 2015. Last visited on 19/09/2016.

[137] T. Flach, N. Mishra, L. Pedrosa, C. Riesz, and R. Govindan, "Carma: Towards personalized automotive tuning," in *Proceedings of the 9th ACM Conference on Embedded Networked Sensor Systems*, (New York, USA), pp. 135–148, ACM, 2011.

[138] R. Ltd, "Cost effective updating of software in cars from IVIs, TCUs and domain controllers to the entire vehicle," white paper, Redbend, Feb. 2015. howpublished=http://www.redbend.com/data/upl/whitepapers/Cost%20Effective%20Updating%20of%20Software%20in%20Cars%20Whitepaper.pdf, Last visited on 25/10/2016.

[139] R. Schmidgall, *Automotive embedded systems software reprogramming.* phdthesis, Brunel University School of Engineering and Design PhD Theses, 2012. howpublished=http://bura.brunel.ac.uk/handle/2438/7070.

[140] ESCRYPT, “Secure SOTA/FOTA management for automotive.” https://www.escrypt.com/fileadmin/escrypt/pdf/ESCRYPT_Software_Updates_OTA.pdf. Last visited on 20/09/2016.

[141] Microchip, “Wi-Fi G demo board users guide.” http://ww1.microchip.com/downloads/en/DeviceDoc/50002147A.pdf, Oct. 2013. Last visited on 22/10/2016.

[142] Microchip, “PIC32MX5XX/6XX/7XX family data sheet.” http://ww1.microchip.com/downloads/en/DeviceDoc/61156H.pdf, Nov. 2012. Last visited on 22/10/2016.

[143] Microchip, “MRF24WG0MA/MB.” http://ww1.microchip.com/downloads/en/DeviceDoc/70686B.pdf, Nov. 2011. Last visited on 22/10/2016.

[144] Microchip, “PIC18F2480/2580/4480/4580 data sheet.” http://ww1.microchip.com/downloads/en/DeviceDoc/39637c.pdf, Aug. 2007. Last visited on 22/10/2016.

[145] Microchip, “MCP2200 breakout module user’s guide.” http://ww1.microchip.com/downloads/en/DeviceDoc/52064A.pdf, Nov. 2012. Last visited on 22/10/2016.

[146] i2cchip, “RealTerm: Serial capture program 2.0.0.70.” https://sourceforge.net/projects/realterm/files/Realterm/2.0.0.70/, Apr. 2012. Last visited on 19/09/2016.

[147] Microchip, “High-speed CAN transceiver.” http://ww1.microchip.com/downloads/en/devicedoc/21667d.pdf, July 2003. Last visited on 22/10/2016.

[148] S. Watson and A. Dehghantanha, “Digital forensics: The missing piece of the internet of things promise,” *Computer Fraud & Security*, vol. 2016, pp. 5–8, June 2016.

[149] S. Al-Kuwari and S. D. Wolthusen, “On the feasibility of carrying out live real-time forensics for modern intelligent vehicles,” in *Forensics in Telecommunications, Information, and Multimedia* (X. Lai, D. Gu, B. Jin, Y. Wang, and H. Li, eds.), vol. 56 of *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*, pp. 207–223, Shanghai, China: Springer, Nov. 11-12 2010.

[150] T. M. Kowalick, "Motor vehicle 'EDR' global standardisation and related issues." http://onlinepubs.trb.org/onlinepubs/UA/111610Kowalick.pdf, Nov. 2010. Last visited on 19/09/2016.

[151] P. Handel, I. Skog, J. Wahlstrom, F. Bonawiede, R. Welch, J. Ohlsson, and M. Ohlsson, "Insurance telematics: Opportunities and challenges with the smartphone solution," *IEEE Intelligent Transportation Systems Magazine*, vol. 6, no. 4, pp. 57–70, 2014.

[152] D. K. Nilsson and U. E. Larson, "Conducting forensic investigations of cyber attacks on automobile in-vehicle networks," in *Proceedings of the 1st international conference on Forensic applications and techniques in telecommunications, information, and multimedia and workshop*, (Brussels, Belgium), p. 8, ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2008.

[153] T. Hoppe, S. Kuhlmann, S. Kiltz, and J. Dittmann, "IT-forensic automotive investigations on the example of route reconstruction on automotive system and communication data," in *Computer Safety, Reliability, and Security* (F. Ortmeier and P. Daniel, eds.), vol. 7612 of *LNCS*, (Magdeburg, Germany), pp. 125–136, Springer Berlin Heidelberg, Sept 25-28 2012.

[154] T. P. forum, "Toyota PRIUS CAN ID." http://www.john2211.nl/forum/ScangaugeII.htm, Apr. 2011. Last visited on 19/09/2016.

[155] X. Solutions, "Auto Care - car maintenance service and gas log." https://itunes.apple.com/gb/app/auto-care-free-car-maintenance/id576958809?mt=8, May 2014. Last visited on 19/09/2016.

[156] AUTOsist, "Car maintenance & gas log - track/manage vehicles." https://itunes.apple.com/gb/app/autosist-car-motorcycle-vehicle/id897916520?mt=8, Oct. 2016. Last visited on 19/09/2016.

[157] J. Monroe, "Car Minder." https://itunes.apple.com/us/app/car-minder-plus-car-maintenance/id310809791?mt=8, Mar. 2014. Last visited on 19/09/2016.

[158] L. Fuelly, "aCar." https://play.google.com/store/apps/details?id=com.zonewalker.acar.pro, Oct. 2015. Last visited on 19/09/2016.

[159] Davag, "My Cars." https://play.google.com/store/apps/details?id=com.aguirre.android.mycar.activity, Oct. 2016. Last visited on 19/09/2016.

[160] Ford, "Ford's vehicle repair and service." http://www.ford.co.uk/OwnerServices/VehicleServiceandRepair/ServicingyourFord/VehicleServicing, 2016. Last visited on 19/09/2016.

[161] U. Toyota Motor Sales, "Toyota service history." http://www.toyota.com/owners/parts-service/history, 2011-2016. Last visited on 19/09/2016.

[162] M. U. Ltd., "Motoriety." http://motoriety.co.uk/, 2015. Last visited on 19/09/2016.

[163] F. Amouzegar and A. Patel, "Vehicle maintenance notification system using RFID technology," *International Journal of Computer Theory and Engineering*, vol. 5, p. 312, Apr. 2013.

[164] H. Hiraoka, N. Iwanami, Y. Fujii, T. Seya, and H. Ishizuka, "Network agents for life cycle support of mechanical parts," in *Environmentally Conscious Design and Inverse Manufacturing, 2003. EcoDesign'03. 2003 3rd International Symposium on*, pp. 61–64, IEEE, IEEE, Dec. 8-11 2003.

[165] G. J. Boss, P. G. Finn, A. H. I. Rick, B. M. O'Connell, J. W. Seaman, and K. R. Walker, "Tracking vehicle maintenance using sensor detection," Nov. 2012.

[166] V. Suresh and V. Nirmalrani, "Android based vehicle diagnostics and early fault estimation system," in *Computation of Power, Energy, Information and Communication (ICCPEIC), 2014 International Conference on*, (Chennai, India), pp. 417–421, IEEE, IEEE, Apr 16-17 2014.

[167] P. Miranda, M. Siekkinen, and H. Waris, "TLS and energy consumption on a mobile device: A measurement study," in *Computers and Communications (ISCC), 2011 IEEE Symposium on*, (Kerkyra, Greece), pp. 983–989, IEEE, IEEE, Jun 28 - Jul 1 2011.