Cryptographic Security for Vehicular Controller Area Networks

Teză destinată obținerii
titlului științific de doctor inginer
la
Universitatea *Politehnica* Timișoara
în domeniul INGINERIA SISTEMELOR
de către

Ing. Pal-Ștefan Murvay

Conducător științific:	prof.univ.dr.ing. Ioan Silea
Referenți științifici:	prof.univ.dr.ing. Toma-Leonida Dragomir
	prof.univ.dr. Dana Petcu
	prof.univ.dr.ing. Alin Suciu

Ziua susținerii tezei: 21.02.2014

Seriile Teze de doctorat ale UPT sunt:

1. Automatică
2. Chimie
3. Energetică
4. Ingineria Chimică
5. Inginerie Civilă
6. Inginerie Electrică
7. Inginerie Electronică și Telecomunicații
8. Inginerie Industrială
9. Inginerie Mecanică
10. Știința Calculatoarelor
11. Știința și Ingineria Materialelor
12. Ingineria sistemelor
13. Inginerie energetică
14. Calculatoare și tehnologia informației
15. Ingineria materialelor
16. Inginerie și Management

Universitatea Politehnica Timișoara a inițiat seriile de mai sus în scopul diseminării expertizei, cunoștințelor și rezultatelor cercetărilor întreprinse în cadrul Școlii doctorale a universității. Seriile conțin, potrivit H.B.Ex.S Nr. 14 / 14.07.2006, tezele de doctorat susținute în universitate începând cu 1 octombrie 2006.

România, 300159 Timișoara, Bd. Republicii 9,
Tel./fax 0256 403823
e-mail: editura@edipol.upt.ro

Cuvânt înainte

Rezultatul unei cercetării relevante trebuie să lărgească sfera cunoașterii precum o galerie nouă adăugată pe harta unei peșteri. Lucrarea de față își propune să facă același lucru reflectând rezultatele pe care le-am obținut în perioada stagiului doctoral desfășurat în departamentului de Automatică și Informatică Aplicată din cadrul Universității Politehnica Timișoara.

Finalizarea acestei lucrări a fost posibilă datorită tuturor celor care m-au îndrumat și sprijinit. În cele ce urmează voi încerca să trec în revistă persoanele care mi-au influențat activitatea desfășurată în postura de doctorand și nu numai.

Domnului prof.dr.ing. Ioan SILEA doresc să-i mulțumesc pentru sprijinul constant și încurajările oferite, în calitate de coordonator, pe întreaga durată a doctoratului.

Încă de la absolvirea facultății formarea mea științifică a fost călăuzită dr.ing. Bogdan GROZA, cel în care am găsit un dascăl și totodată un prieten. El m-a convins să-mi continui studiile și mi-a stârnit apetitul pentru cercetare în domeniul securității suficient încat să mă înscriu la doctorat. Pe lângă noțiunile de securitate și criptografie pe care mi le-a transmis, m-a învățat despre etica științifică și lumea insectelor. Pentru toate acestea îi mulțumesc.

În forma finală a lucrărilor publicate în cei trei ani de doctorat se reflectă și sfaturile oferite de recenzorii anonimi cărora le sunt recunoscător pentru efortul depus.

La fel de important pentru realizarea acestei teze, a fost și faptul că am avut alături de mine familia și prietenii. Părinții mi-au îndrumat primii pași în viață. În ei am găsit modelul demn de urmat fără de care nu aș fi ajuns omul care scrie aceste rânduri. Iubita mea Alina mi-a ascultat cu răbdare trăncăneala în momentele de entuziasm sau frustrare și mi-a oferit înțelegere și afecțiune chiar și în clipele în care nu i-am acordat suficientă atenție. Sora mea, bunicii și prietenii mi-au alimentat și ei ambiția de a merge mai departe. Tuturor le mulțumesc pentru că au fost acolo și îmi doresc să rămână mereu aproape.

Timișoara, Februarie 2014 Pal-Ștefan MURVAY

This work was partially supported by the strategic grant POSDRU 107/1.5/S/77265, inside POSDRU Romania 2007-2013 co-financed by the European Social Fund - Investing in People.

In memoriam I. TOFAN et P. MURVAY.

Nam quicumque crediderunt in me.

Murvay, Pal-Ștefan

Cryptographic Security for Vehicular Controller Area Networks

Teze de doctorat ale UPT, Seria 12, Nr. 10, Editura Politehnica, 2014, 126 pagini, 39 figuri, 23 tabele.

ISSN: 2068-7990
ISBN: 978-606-554-774-2

Cuvinte cheie: automotive, controller area network, protocol de autentificare, securitate

Rezumat:

Automobilele moderne sunt dotate cu subsiteme din ce în ce mai complexe menite să îmbunătățească siguranța și confortul utilizatorilor. Creșterea complexității subsistemelor, aduce însă și vulnerabilități noi crescând numărul suprafețelor de atac în sistem precum și cel al modurilor în care un adversar poate să le exploateze (lucru demonstrat de o serie de articole recente). În mare parte, atacurile vizează infiltrarea magistralelor interne de comunicare ale automobilelor care, în lipsa unor măsuri de protecție, pot fi folosite pentru a manipula funcționarea sistemului.

Lucrarea de față propune și analizează o serie de soluții pentru asigurarea autentificării în comunicarea pe rețele Controller Area Network (CAN), acestea fiind cele mai des utilizate rețele în aplicații automotive. Constrângerile dispozitivelor embedded utilizate în autovehicule exclud asigurarea autenticității prin utilizarea unor soluții clasice din criptografia cu cheie publică (de ex. semnături digitale). Soluțiile propuse în lucrare au fost proiectate pentru a funcționa în prezența acestor constrângeri și se încadrează în două categorii: protocoale de autentificare implementate la nivelul aplicației și identificarea folosind semnale de la nivelul fizic.

În ceea ce privește autentificarea la nivelul de aplicație, soluțiile propuse au la bază diferite concepte de utilizare a primitivelor criptografice simetrice (cheie secretă) urmărind reducerea costului computațional dar și a lățimii de bandă. Binecunoscutul protocol TESLA respectiv semnăturile one-time sunt utilizate în două protocoale, ambele bazate pe folosirea lanțurilor one-way. O a treia soluție, protocolul LiBrA, propune asigurarea autenticității prin utilizarea a două paradigme: partajarea de chei în grupuri de noduri și mixarea MAC-urilor.

La nivelul fizic, teza propune și o abordare inovatoare pentru asigurarea autenticității pe baza caracteristicilor unice ale semnalelor CAN generate de fiecare nod (transceiver). Rezultatele experimentale arată că este fezabilă identificarea nodurilor CAN cu un nivel ridicat de certitudine.

Contents

Nomenclature

Acronyms

AES	Advanced Encryption Standard
ASIC	Application Specific Integrated Circuit
CAN	Controller Area Network
CAN FD	CAN with Flexible Data-Rate
CRC	Cyclic Redundancy Check
DES	Data Encryption Standard
DoS	Denial of Services
ECU	Electronic Control Unit
EDR	Event Data Recorder
FPGA	Field Programmable Gate Array
LIN	Local Interconnect Network
MAC	Message Authentication Code
MD5	Message Digest (hash function)
MOST	Media Oriented Systems Transport
NCS	Networked Control System
NIST	National Institute of Standards and Technology
PLC	Programmable Logic Controller
RFID	Radio-Frequency Identification
SHA	Secure Hash Algorithm family of hash functions
TESLA	Timed Efficient Stream Loss-tolerant Authentication protocol

List of Figures

List of Tables

1 Introduction

In our fast paced world, intelligent electronics are continuously evolving to make our day-to-day life easier by bringing improvements in all areas of activity. Automobiles have become ubiquitous and just like any other devices, have benefited from the enhancements brought by electronics. Car manufacturers work to offer better performance as well as improved comfort and safety aiming at an overall better usage experience for vehicle owners. This makes automotive systems more and more complex. Individual functionalities available in today's car are usually provided by separate electronic control units (ECUs) which have to interact in order to fulfill their purpose. However, the increasing number of functionalities and connectivity options bring vulnerabilities that can be exploited to cause damage in the absence of security mechanisms.

1.1 Problem statement

Due to the growing complexity of modern automobiles, vehicular communication became an essential topic in the automotive industry. A wide range of solutions (wired buses as well as wireless approaches) were adopted to fulfill the communication needs of the automotive systems. The rapid evolution of these came at the cost of introducing a series of new possible attack surfaces, e.g., multimedia devices, wireless channels such as Bluetooth.

Until recently, the security of vehicular systems was not considered to be a major concern. This, however, changed as a series of attacks were reported by the scientific community. The increasing number of reported vulnerabilities and their impact on driver/passenger safety underline the importance of the subject and call for the development of secure communication inside vehicles.

Assuring the security of modern cars is not a straight-forward task. The development process involves the evaluation of attacker capabilities and a thorough analysis of potential vulnerabilities. This can be achieved by including penetration testing as a step of the development. Security mechanisms should be then employed to alleviate the identified vulnerabilities and vulnerable communication channels need to be secured to prevent malicious actions. However, the applied security techniques must comply with a series of constraints related to their implementation cost and effect on the system reliability. After security has been implemented the bus, CPU and memory loads should not be significantly increased.

Buses used for building in-vehicle networks were designed to offer reliable communication, thus they are fitted with efficient mechanisms for error detection and

error recovery. However, they are lacking support for assuring basic security objectives. Obviously, the best solution, from the security point of view, would be to devise a dedicated bus with built-in mechanisms for assuring security at the network layer rather than by the application layer. As such an approach would involve longer time for introduction and high costs (for creating specifications of the new protocol, producing corresponding transceivers and testing them properly) it is not considered as a viable option in the short term. In order to keep costs down and avoid the long process of adopting new protocols, currently used buses could be retrofitted with security mechanisms implemented at the application level.

Implementing security on microcontrollers used to build ECUs without increasing costs can be challenging as they are resource-constrained devices in what regards available memory and maximum working frequency. Due to low tolerance margins most of the automotive microcontrollers employed by the industry would likely support only algorithms based on lightweight cryptography. Employing microcontrollers enhanced with cryptographic co-processors can enhance the use of public key cryptography but this might not be acceptable as automotive component manufacturers try to keep production costs low to provide affordable end-products. Work has been done in developing specialized hardware security modules for vehicular systems [124] which might reduce costs if such modules would get to be used at a large scale in the automotive industry. However, it is not sure if such a solution will be adopted by the industry.

Secure communication is a crucial component needed for providing safety in automotive systems. However, assuring it is a difficult task as any solution has to fulfill a series of intrinsic constraints (related to the device itself) and a series of extrinsic constraints (related to the system), e.g., delays.

In this context, the purpose of this thesis is to address the subject of authenticated in-vehicle communication.

1.2 Thesis objectives

In general terms, the main goal of this thesis was to bring new and relevant contributions in the field of in-vehicle security. The contributions presented in this thesis correspond to several different research directions which will be described in what follows.

The study of reported vulnerabilities of in-vehicle networks was set as a first objective. Attacker models were first studied to create a sketch of the adversaries that have to be counteracted in the given context. Next, a comprehensive survey on reported attacks and attack surfaces was done to emphasize the relevance of the rest of objectives.

As the devices commonly used in automotive applications are known to come with a series of constraints, to depict them we start from analyzing the capabilities of these devices. Several automotive-type microcontrollers were selected to depict the computational performance achievable using this type of devices for implementing

security. On-chip features such as parallelism and hardware acceleration present on the employed platforms are used to present performance enhancements.

The third objective was to identify solutions for providing authentication on CAN buses at the application layer. Both existing approaches and new ones are investigated and experimental results are presented for the designed solutions. The effect of introducing these authentication protocols to CAN communication was also in our focus as it is of significant importance for the correct functioning of the vehicle system.

Finally, since the results achievable by using the application layer are bounded by microcontroller's performance and bandwidth, we investigate the possibilities of achieving authentication at the physical layer. This could be done by identifying unique characteristics in the physical signal produced by each transceiver.

1.3 Thesis outline

The thesis starts by presenting the background and motivation of this work in Chapter 2. Here, details about in-vehicle networks and buses employed in automotive applications are first given before proceeding to a comprehensive survey of adversary models and attacks reported in this type of networks. The remainder of the thesis holds the main contributions of the author and the conclusions. A performance analysis of automotive-specific microcontrollers along with performance improvements achievable when using chip-specific features are presented in Chapter 3. Chapter 4 presents three categories of authentication protocols for CAN networks and the performance achievable when using them. Next, a physical layer approach to achieving authentication of CAN sender nodes is described in Chapter 5. Finally, Chapter 6 is dedicated to the conclusions of this work.

2 Background and motivation

2.1 Controller Area Network

2.1.1 In-vehicle networks

Sensors, actuators and ECUs inside a car need to communicate to fulfill their purpose. The initial approach was to provide a direct wired connection between each two components that have to exchange information. However, since the introduction of electronics in the automotive domain, the number of wired connections needed inside a car quickly increased making it clear that a more efficient solution was needed. Therefore, a more structured approach was adopted for building the in-vehicle network by using dedicated buses. The complexity of the in-vehicle network also increased due to the growing number of features made available inside the car. A car being built today can have an average of 20 network nodes and this average is estimated to increase to 25 just by the end of 2014 [18]. In high-end cars the number of network nodes can increase to 80-100 such as the case of the Audi A8 which contains 100 ECUs [6].The architecture of in-vehicle networks used in nowadays cars reunites a wide range of wired communication buses and even wireless channels that are utilized to fulfill the industry's needs.

Figure 2.1 depicts the wired (wireless channels not included) network topology of an Audi A8 adapted after [6]. As most of in-vehicle networks, it is divided in several sub-networks dedicated to different categories of vehicle systems such as: powertrain, infotainment or comfort. These main sub-networks are connected to a central gateway along with the diagnostics interface. While the network complexity will vary from low-end to high-end vehicles, this example illustrates the range of buses that can be employed in a single automotive network. Some communication buses were specially designed to be used for in-vehicle communication while others were "borrowed" from other domains such as avionics. Some of the most commonly used vehicle buses are presented in what follows.

Local Interconnect Network (*LIN*) is a low cost, serial broadcast bus with a single master/multiple slave network topology. With a maximum data rate of 19.2 kbaud and single wire operation capability, this bus is used to bridge communication between ECUs and peripherals (sensors and actuators) which are usually implemented as ASICs.

Controller Area Network (*CAN*) is a differential serial broadcast bus which allows communication speeds of up to 1 Mbaud. More details about this bus will be given in the next section.

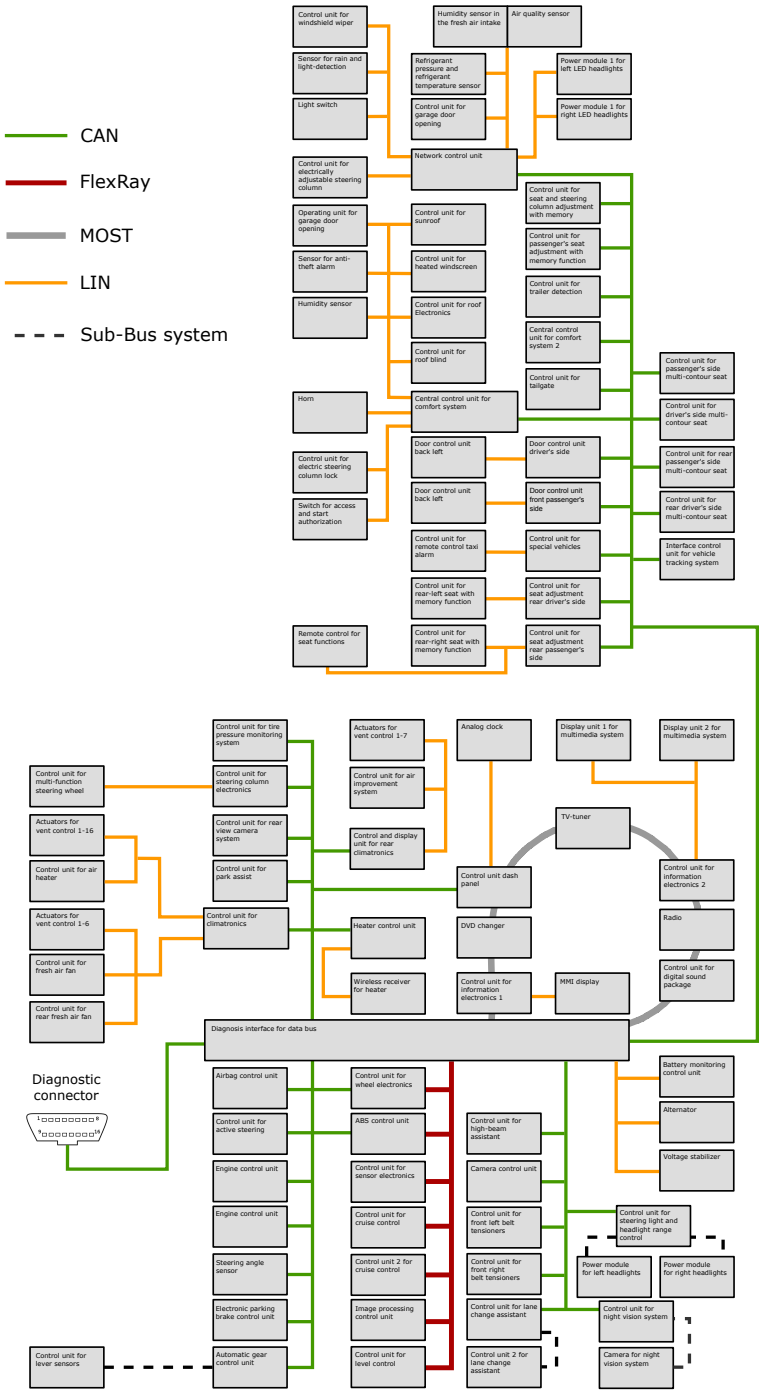


Figure 2.1: Topology of the Audi A8 (2010) in-vehicle network adapted after [6]

FlexRay was designed by a consortium comprised of automotive manufacturers and suppliers as a high speed (up to 10 MBaud), fault tolerant, serial communication bus that can hold a maximum payload of 254 bytes. FlexRay provides deterministic behavior along with time-triggered and event-triggered capabilities. The downside of using FlexRay is represented by its increased cost and complexity.

Media Oriented Systems Transport (*MOST*), as suggested by its name, is a bus designed for multimedia applications which can assure bit rates up to 150 Mbaud in latest generation devices.

As the complexity of in-vehicle networks is ever-increasing along with the need for higher bandwidth, introducing Ethernet in automotive applications is starting to be considered as an alternative [18].

Wireless in-vehicle communication is also being employed for applications such as tire pressure monitoring, remote keyless entry, immobilizers or multimedia. Depending on the features available, several wireless communication channels may be present in an automobile.

Even though a great variety of network types is available for in-vehicle communication, CAN is currently the most widely used in automobiles and it is likely that its use will remain widely spread due to its reliability and reduced cost. Therefore, the results presented in this thesis are focused on CAN communication. The next section holds a more detailed description of the CAN protocol.

2.1.2 CAN technical details

Since its first official release, in 1986 by Robert Bosch GmbH., and the production of the first controller chips, CAN was adopted by all major car manufacturers and became a standard for in-vehicle communication in the automotive industry. Version 2.0 of the CAN specification [107] was published by Bosch in 1991. Starting from 2003 a multipart ISO standard (ISO 11898-1 through 5 [64]) describes the physical and data link layer of CAN.

CAN was designed as a multi-master broadcast serial bus with efficient error detection and arbitration mechanisms. Signals on the bus can have one of two values: *dominant* or *recessive*. The *dominant* level is represented by a logical '0' while a logical '1' will be used to specify a *recessive* level. CAN is a message oriented bus which means that each message has an identifier assigned to it that is used to define its priority. When 2 or more nodes begin to transmit a message at the same time, a bit by bit comparison is done (this is the arbitration phase). A *dominant* bit will always win arbitration, therefore, when a node puts a *recessive* bit on the bus and reads back a *dominant* bit it will stop sending and wait until the bus is idle again. As a consequence of this mechanism, lower message IDs will always win arbitration.

The efficient error detection mechanisms implemented in CAN are based on message monitoring, cyclic redundancy check (CRC), bit stuffing and message frame checks. The probability of an undetected error on CAN is extremely low, informally one undetected error occurs at about one thousand years for each vehicle that operates

eight hours a day with an error each 0.7 seconds. Upon detection, an error is signaled on the bus by the node that detects it so that the fault confinement mechanisms can be applied. To assure that the bus communication is not disturbed by defective nodes, CAN has built-in mechanisms for switching off faulty nodes.

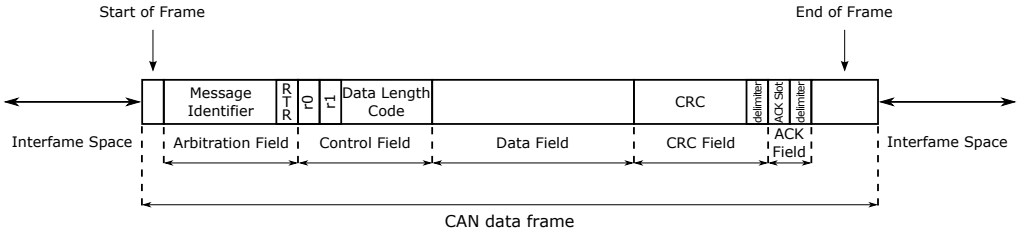


Figure 2.2: Structure of a CAN data frame

The structure of a CAN data frame is depicted in Figure 2.2. It begins with a *dominant* bit marking the start of the frame followed by the arbitration field which consists of the message identifier (11 bits in standard frames and 29 bits for extended frames) and a *dominant* RTR (Remote Transmission Request) bit. The control field holding two bits reserved for further extensions and a 4 bit representation of the actual message length comes next followed by a maximum of 64 bits of data. A 15 bit CRC is appended next followed by a *recessive* CRC delimiter and a two bit ACK field. The frame end is marked by a sequence of 7 *recessive* bits.

The typical CAN topology consists of a two wire differential bus as depicted in Figure 2.3. A low value resistor (e.g. 120 ohms) is connected between the two bus wires, called CAN-Low (CAN-) and CAN-High (CAN+), to alleviate noise. In fault-tolerant/low-speed CAN networks, each device on the line needs a terminator resistor for each line and the communication can be switched to single-wire mode in the presence of faults.

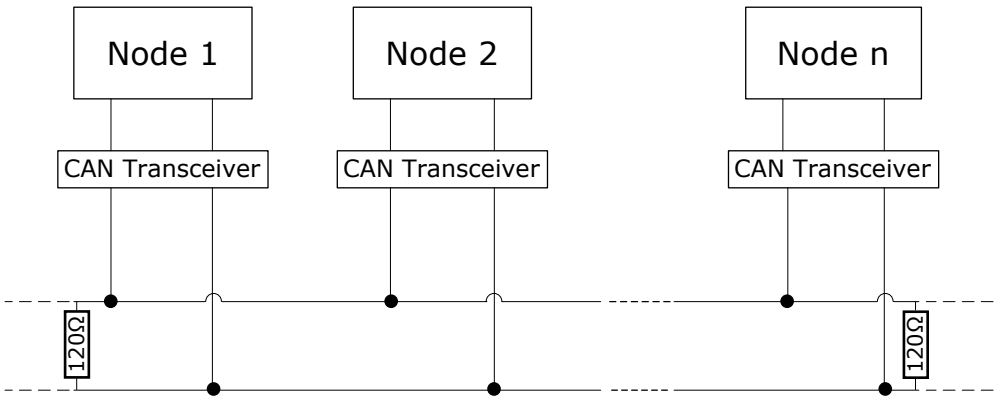


Figure 2.3: Typical CAN bus topology

2.1.3 CAN in networked control systems

Systems in which control loops are closed through a network are classified as networked control systems (NCS). The defining characteristic of an NCS is that information exchanged among its components (sensors, actuators, controllers, etc.) is sent through a network shared with nodes that are not necessarily part of the control system. The communication channels used to provide real-time communication in these control systems are implemented by industrial network protocols known as fieldbuses. CAN is one of the fieldbus standards commonly used in various NCS applications [50]. CAN may not offer the best data rate among the commonly employed fieldbuses but it is preferred in some applications due to its reliability. An extensive performance evaluation of CAN along other buses regarding their use in control applications can be found in [74].

In 1992 an organization named CAN in Automation (CiA) was founded with the intent of promoting CAN and its usage in various fields. About 560 companies with different fields of activity are currently members of this organization pointing out the multiple applications of CAN. Automotive control systems are probably the main application area for CAN buses. However, CAN is used in a wide range of other control systems such as home automation [73], [112] and industrial applications [72], [3]. Some example of relevant applications of CAN in networked control systems will be presented in what follows.

In [72] Lawrence presents detailed examples of CAN application in general such as in an universal industrial process control systems or more specific examples such as textile, construction or mining machines.

Control systems for mobile robots were built on CAN-based architectures in the case of wheeled robots [16] as well as for humanoid robots [66]. CAN-based control is also employed in systems with robotic arms that perform complex tasks such as surgery [128]. In all these cases CAN was preferred due to its high reliability.

Subsea instrumentation used in the offshore oil and gas industry has to provide reliable control of the production process. Subsea production control systems are designed to operate on the seabed, at depths in excess of 1000 meters, without maintenance for periods between 25 and 30 years. The remote environment in which these systems have to function require the usage of reliable components to prevent faults that could lead to financial loses, human injuries or environmental disasters. Due to its efficient error management, CAN was adopted as an interface standard in subsea control applications [26]. More specifically, the fault tolerant version of CAN was considered as it is capable of providing operation even in the presence of a partial system fault. CAN's role in these systems is to transport information between sensors and subsea control modules.

Another use case for CAN is in marine automation systems which have complex structures usually partitioned into several process segments [31]. Each process segment is dedicated to the control of a certain subsystem like power management, pumps and valve or auxiliary systems. The process data transfer within each subsys-

tem is done over a local communication system implemented using CAN. The global process bus that interconnects main process controllers of each automation subsystem is also implemented on CAN. The communication protocol employed in these systems is an extension of the CANopen standard, a higher layer protocol based on CAN physical and data link layer [30].

As different applications have specific protocol requirements, other higher layer protocols were developed on top of CAN. One of these is DeviceNet [95] which was used in various industrial control applications. DeviceNet is mainly used in factory automation applications, e.g. the control of composite wood panels manufacturing process presented in [68].

Process control systems also bring benefits to the food industry. For example the conservation and maturation of fruits has to be done in controlled environments. In [78] a distributed control system for citric fruit conservation and maturation is presented. This system is equipped with a main CAN backbone that connects several nodes, each of which is in charge of a cold store chamber. Sensors and actuators are connected to the node controlling a each cold store via a CAN subnetwork. The system is accessible through the internet to allow remote monitoring and control of the running processes.

CAN connectivity is available in various off-the-shelf automation solutions from manufacturers such as Siemens, dSPACE, National Instruments, etc. emphasizing the relevance of this bus system in the process control industry.

While this thesis studies the case of CAN usage in automotive control systems, security mechanisms may be required in other industrial process control applications where CAN is employed. Also, recent incidents of international level, such as the Stuxnet worm, have shown that industrial control systems are not as isolated as once thought and can become vulnerable targets [32]. Stuxnet was designed to infiltrate industrial control systems and reprogram Siemens programmable logic controllers (PLCs) to sabotage an uranium enrichment system by slowing down or speeding up centrifuge motors to different rates at different times [81]. The targeted system used Profibus communication which, like CAN, provides only a CRC based mechanism for assuring data integrity without assuring source authentication. However, Siemens PLCs can also use CAN communication modules making this kind of attacks a real threat also in CAN based systems.

2.1.4 CAN security

No intrinsic security mechanism was included in the CAN protocol specification apart from the 15 bit CRC that only assures data integrity. This makes CAN vulnerable to simple sniff and replay attacks. Thus, if the authenticity or confidentiality of the transmitted data has to be assured this must be implemented at a higher level.

The design of the CAN arbitration mechanism makes it susceptible to DoS attacks. As stated before, messages with lower IDs have higher priority. Therefore, a malicious node could continuously send a message with id $0x000$ to prevent other nodes

gaining access to the bus. In addition, by sending several well-directed error flags, an attacker could disconnect target CAN nodes by using CAN automatic fault localization and confinement mechanisms [125]. These problems have no solution except from redesigning the bus architecture. Nevertheless, since all these cause a DoS which is the logical equivalent of cutting the wires they are not considered as relevant as the sniff and replay attacks that can insert adversary packages at will.

2.1.5 CAN with Flexible Data-Rate

As a result of the increasing demand for bandwidth there were many attempts to improve and extend the CAN protocol. One approach employed was to use a star topology [20] while other research was focused on using higher bit rates for the data segment of the CAN frame [130]. Recently, new efforts have been made to develop and standardize a new CAN extension that can provide data rates higher than 1 Mbaud and payloads up to 64 bytes. The new specification, called CAN FD (CAN with Flexible Data-Rate) [108], states that messages in the CAN frame format and messages in CAN FD frame format can coexist in the same network. This means that CAN FD nodes will be able to receive messages from CAN nodes while CAN nodes would have to be passively disconnected during FD communication because they would otherwise report errors on the bus.

The CAN FD frame structure introduces three new control bits: EDL (Extended Data Length), BRS (Bit Rate Switch) and ESI (Error State Indicator). A *recessive* EDL bit denotes a CAN FD frame while for a standard CAN frame this bit is set to *dominant*. A *recessive* BRS bit switches to the higher bit rate when sending the data field. The ESI bit was introduced for signaling an error state. The message length field is composed of 4 bytes, as in the case of standard CAN frames, but can specify data fields longer than 8 bytes: 12, 16, 20, 24, 32, 48 and 64 bytes. The size of the CRC field varies depending on the length of the data field: 15-bit CRC for standard CAN frames, 17-bit CRC for a data field up to 16 bytes in length and 21-bit CRC for a data field longer than sixteen bytes.

The higher bit rate is only available during the data phase, i.e. the frame space bounded by the BRS and CRC delimiter bits. Bit rates of up to 15 MBit/s were achieved in the data phase during the development of CAN FD using FPGA implementations [53]. However, it is not expected for such data rates to be achieved in automotive conditions using existing transceivers. CAN FD is rather viewed as an intermediary solution between CAN and FlexRay.

While the automotive industry manifests an active interest in CAN FD it is not yet clear how fast will this bus technology be adopted.

2.2 Security issues for in-vehicle networks

Many new features were introduced to modern automobiles with the purpose of increasing reliability, safety and user experience, e.g.: passive keyless entry, immobi-

lizers, multimedia, tire pressure monitoring, telematics, vehicle diagnostics. However, recent research [23, 36, 70, 110] has shown that some of these systems have vulnerabilities that can be exploited by parties looking for financial gains or even having the malicious intent to cause accidents. In-vehicle networks evolved as a consequence of the continuously increasing system complexity. Unfortunately, once infiltrated, these networks provide little to no protection against attacks [23, 55, 56, 69, 70, 110]. With the increase in the complexity of vehicular embedded systems comes an increase in the overall code size and, inevitably, a greater value of the intellectual property [103]. This is why vehicle manufacturers have focused on devising methods for preventing unauthorized software changes in the marketed components. However, it is common knowledge that chip tuning services can be bought from unauthorized individuals for the majority of car makes and models.

The impact that existing vulnerabilities may have when they are used by attackers, to endanger driver safety or to induce financial losses, underline the necessity of increasing the focus on the security of these systems. In order to counteract possible attacks it is essential to have a clear idea on who the adversaries are, what they can do and what is their purpose. The answer to these questions can be the result of a design time attack analysis as suggested in [43].

2.2.1 Attacker model

Various attacker models have been presented as part of published work in the area of automotive security. However, to the best of our knowledge, there exists no comprehensive paper on this subject. Most of the models are either too general, such as the Dolev-Yao attacker [27], and can be applied in any branch of information security, either too limited fitting only a particular case. We present some of these models next.

Nilsson and Larson [93], [94] use a model that gives an attacker the ability to access the in-vehicle network and to read, spoof, drop, modify, flood, steal, and replay messages. Koscher et al. [70] consider two categories of attackers. On one side there are attackers that have physical access to the vehicle for a long enough period to introduce a malicious element in the vehicle network or to compromise one of the existing network nodes. On the other side they consider attackers that use the available in-vehicle wireless communication channels as an attack surface. The same research group later performs a threat modeling on automotive attack surfaces [23] and identifies several entry points.

Wolf et al. identified four groups of attackers in the automotive area [126]: thieves, car owners, garage personnel and a fourth group consisting of organized crime, competitors or counterfeiters. Their risk assessment points to the last category as being the most powerful and dangerous one. In [55], Hoppe and Dittmann propose an adapted CERT taxonomy for use in the automotive environment. This taxonomy contains 11 kinds of attackers: hackers, spies, terrorists, corporate raiders, professional criminals, vandals, voyeurs, security scans, tuners, thieves and competing manufac-

turers. They also mention a series of tools, vulnerabilities, actions, possible targets, results and objectives that can be used to classify an incident.

Another approach, employed by Müter and Freiling [89], is to define an attacker by selecting relevant security components in the vehicular system model. They consider that the strengths and abilities of an attacker are specified by the selection of component properties that are relevant for this attacker.

In order to have an understanding of the attacker behavior we first have to identify the assets of the automotive system. Probably the most commonly affected asset is *the car* itself along with valuable *objects left inside the car*. Systems like smart keys, remote keyless entry and immobilizers were introduced to protect these assets but they proved to be insecure [79], [33]. Even *car components*, especially the expensive electronic parts, are a significant target for thieves. On the other hand, *human safety* is the most important asset that can be targeted by an attack. *Privacy* can also be an issue for example when using the integrated hands free functionality via bluetooth. The great variety of counterfeit vehicle parts available on the market point to another asset, namely *intellectual property*. The discovery of security flaws means bad publicity for the involved manufacturer, therefore we consider *manufacturer reputation* as being another important asset.

We define the attacker as being a malicious entity characterized by a set of four features: *motivation* (M), *access level* (A), *capability* (C) and *purpose* (P). By *motivation* we refer to the reason or reasons that drive an attacker to perform a malicious action. The *access level* feature illustrates the category in which the attacker fits based on the access he has to the target vehicle while the *capability* considers everything related to the level of knowledge, skills, technical competences and technology available to the attacker. Finally, the *purpose* represents the effect expected by the attacker as a result to his actions. Each features can be described by one or more items from a set of possible values. We give some examples below:

- $M = \{financial, reputation, revenge, scientific, ...\}$
- $A = \{vehicle\ owner, insider, outsider, ...\}$
- $C = \{programming, electronics, well\ equipped, ...\}$
- $P = \{car\ theft, component\ theft, cause\ accident, ...\}$

These features can be further detailed to fit the desired level of abstraction. For the remainder of this work we will consider an attacker which has full access to the vehicle with all its components and also possible knowledge about the security protocols employed. This attacker also has relevant technical expertise and is equipped with state-of-the-art equipment which would allow him to perform complex attacks.

2.2.2 Attacks

In recent years, the scientific community manifested an increasing interest on the topic of automotive security. Some of the research done in this area was focused on

finding vulnerabilities that can enable malicious actions to be performed on vehicle systems. As a result, it was shown that in most cases security mechanisms are either missing or susceptible to attacks. Attacks aimed at electronic components from virtually all vehicle functional domains using various attack surfaces such as the OBD port, wireless communication channels and multimedia devices were reported. The results published shed a light on the techniques currently available to malicious parties such as car thieves or persons selling unauthorized chip tuning services.

Table 2.1 holds a compilation of the attacks reported in recent scientific literature organized by the affected functional domains as described in [113]. For each table entry, the affected system, attack effect, attack surface and required access are given. While this is not meant to be an exhaustive study of recently reported attacks, it is comprehensive enough to illustrate vulnerabilities present in contemporary automobiles.

As the table illustrates, the CAN bus was the most used attack surfaces in the security analyses published so far. This implies that physical access to the OBD port or directly to the CAN bus lines must be achieved. If the attacker is the owner himself then he has unlimited access to the vehicle. We find this prerequisite to be achievable also for parties other than the owner as there are various situations in which this access can be gained in his absence, e.g. auto service, car wash, valet parking, etc. Wireless communication channels are also employed to launch a considerable number of attacks with significant impacts. However, as wired buses are still prevalent in automotive applications we focus our attention to devising methods of protecting against attacks on these surfaces, and more specifically the CAN bus.

Affected system	**Attack**	**Attack surface**	**Required access**
Power Train (longitudinal propulsion: engine, transmission...)			
Engine	Increase Idle RPM [70]	CAN bus	OBD port - diagnostics mode
	Temporary RPM increase [70]	CAN bus	OBD port - diagnostics mode
	Initiate crankshaft re-learn (disturbs engine timings) [70]	CAN bus	OBD port - diagnostics mode
	Disable cylinders [70]	CAN bus	OBD port - diagnostics mode
	Kill Engine [70]	CAN bus	OBD port - diagnostics mode
	Grind starter [70]	CAN bus	OBD port - diagnostics mode
	Remote car start [70]	CAN bus	OBD port
	Cannot turn on (DoS to/from BCM) [70]	CAN bus	OBD port
	Cannot turn off (while turned on, cause BCM to activate ignition output) [70]	CAN bus	OBD port
	Accelerate for short periods of time [83]	CAN bus	Internal CAN bus
	Kill engine at any speed [83]	CAN bus	OBD port - diagnostics mode
Chassis (wheels and their relative position and movement: steering, braking...)			
Power steering	Disable [70]	CAN bus	OBD port - diagnostics mode
	DoS attack - limits steering capability [83]	CAN bus	OBD port
	Steer the wheels to any given position [83]	CAN bus	OBD port
Parking Assistance	Induce sporadic wheel jolts [83]	CAN bus	OBD port
Lane Keep Assistance	Turn the wheel while driving [83]	CAN bus	OBD port

Brakes	Disable [70]	CAN bus	OBD port - diagnostics mode
	Engage front left [70]	CAN bus	OBD port - diagnostics mode
	Engage front right, unlock front left [70]	CAN bus	OBD port - diagnostics mode
	Unevenly engage right brakes [70]	CAN bus	OBD port - diagnostics mode
	Release brakes (prevent braking) [70]	CAN bus	OBD port - diagnostics mode
	Engage brakes while the car is stopped [83]	CAN bus	OBD port - diagnostics mode
	Disable while running at low speeds [83]	CAN bus	OBD port - diagnostics mode
Pre-Collision System	Induce autobraking [83]	CAN bus	OBD port
	Engage seat belt pre-tightening motor [83]	CAN bus	OBD port - diagnostics mode
TPMS	Tracking automobiles by sensor IDs [110]	RF	Close proximity (6-40m)
	Packet spoofing from a neighbor car - fake low tire pressure warning [110]	RF	Close proximity (6-40m)
	Battery drain [110]	RF	Close proximity (6-40m)
	Crash TPMS ECU - repeated spoofing [110]	RF	Close proximity (6-40m)
Body (entities not related to vehicle dynamics: wipers, lighting, window lifter, air conditioning, seats...)			
Electric window lift	Open window - sniff and replay attack (simulation CANoe) [55]	CAN bus	Internal CAN bus
	DoS - for each command send opposite command message [56]	CAN bus	Internal CAN bus
	Disable window relays [70]	CAN bus	OBD port
Windshield Wipers	Turn wipers on continuously [70]	CAN bus	OBD port
	Turn fluid shot continuously [70]	CAN bus	OBD port
	Force wipers off & shots fluid continuously [70]	CAN bus	OBD port
Lights	All lights off (brake and auxiliary) [70]	CAN bus	OBD port
	All auxiliary lights off [70]	CAN bus	OBD port
	Disable headlights in auto light control [70]	CAN bus	OBD port
	Turn headlights on or off while in auto light control [83]	CAN bus	OBD port - diagnostics mode
Dome light	Control brightness [70]	CAN bus	OBD port
Trunk door	Pops open [70]	CAN bus	OBD port
Doors	Unlock all (while at speed) [70]	CAN bus	OBD port
	Lock/Unlock car [70]	CAN bus	OBD port
	Continuously activate lock relay [70]	CAN bus	OBD port
	Lock/unlock all while driving [83]	CAN bus	OBD port - diagnostics mode
Horn	Activates permanently [70]	CAN bus	OBD port
	Change Frequency [70]	CAN bus	OBD port
	Turn horn on and off [83]	CAN bus	OBD port - diagnostics mode
Instruments	Control brightness [70]	CAN bus	OBD port
	Falsify speedometer reading [70] [83]	CAN bus	OBD port
	Speedometer drops to 0 (DoS to/from ECM) [70]	CAN bus	OBD port
	Panel freezes (DoS to/fromBCM) [70]	CAN bus	OBD port
	Falsify fuel level [70]	CAN bus	OBD port
	Control various fields on the dashboard [54]	CAN bus	OBD port
	Force odometer value increase [83]	CAN bus	OBD port
	Falsify fuel level [83]	CAN bus	OBD port - diagnostics mode
Smart Junction Box	Shut down causing several systems (lights, radio, HVAC etc.) to stop [83]	CAN bus	OBD port - diagnostics mode
	Start reprogramming causing interior lights to flash [83]	CAN bus	OBD port - diagnostics mode
Remote disable system	Disable cars and sound horn continuously [80]	Remote disable system	Main remote access system

Remote Keyless Entry	Breaking KeeLoq authentication by key recovery - cryptanalysis: known plaintexts, slide attack & meet-in-the-middle [60]	key RF	Within key RF range
	Breaking KeeLoq authentication by key recovery (both remote transmitter and manufacturer key) - side channel: DPA, SPA [97]	key RF	Within key RF range
	Jamming attack - lock signal is jammed and car remains open [36]	key RF	Within car RF range
	Replay attack - unlock message is recorded and replayed [36]	key RF	Within key/car RF range
Passive Keyless Entry	Wired relay attack - open car door & start engine [36]	key RF	Within key/car RF range
	Wireless relay attack - open car door & start engine [36]	key RF	Within key/car RF range
Immobilizer	Tracking of the key fob using Atmel immobilizer protocol stack [118]	RFID	Within key RF range
	DoS Atmel immobilizer protocol stack - Overwrite keys in open and secure mode => key will not work with car [118]	RFID	Within key RF range
	Relay attack on Atmel immobilizer protocol stack [118]	RFID	Within key/car RF range
	Replay attack on authentication for keys based on Atmel immobilizer protocol stack [118]	RFID	Within key/car RF range
	Spoofing attack to lock the EEPROM in Atmel immobilizer protocol stack [118]	RFID	Within key/car RF range
	Retrieve secret key and start engine for Hitag2-based transponders [121]	RFID	Within key/car RF range
Car Alarm	Honk horn [70]	CAN bus	OBD port
Key lock	Disable relays [70]	CAN bus	OBD port
HVAC	Turn fans, A/C or heat on/off [70]	CAN bus	OBD port
Multimedia, Telematics and HMI (information exchange: display, switches, radio, navigation, internet...)			
Media player	Modify song title tags to fake a warning text on the display. Warning sounds can be also added [56]	CD	Plant altered CD
	CD-based firmware update [23]	CD	Plant altered CD
	Play special song to send CAN messages [23]	CD	Plant altered CD
Navigation system	Inject fake RDS-TMC traffic info [10]	RDS-TMC	Reach car radio range
	Manipulate direction reading [83]	CAN bus	OBD port
Radio	Increase volume [70]	CAN bus	OBD port
	Change display [70]	CAN bus	OBD port
	Ticking sound [70]	CAN bus	OBD port
DIC	Change display of Driver Info Center [70]	CAN bus	OBD port
Telematics	Buffer overflow with paired Android phone and Trojan app [23]	Bluetooth	Within Bluetooth range
	Sniff MAC address, brute force PIN, buffer overflow [23]	Bluetooth	Within Bluetooth range
	Call car, authentication exploit, buffer overflow (using laptop) [23]	Cellular	Achievable remotely
	Call car, authentication exploit, buffer overflow (using iPod with exploit audio file, earphones and a telephone) [23]	Cellular	Achievable remotely
Digital video recorder on police cars	Tap into live feeds from police cars and control individual hard drives [80]	FTP, telnet	Achievable remotely

Active/Passive Safety (airbags, warnings, seat belt, ABS, ESP, cruise control...)			
Airbag	Suppress missing airbag warnings by emulating its function with another board [56]	CAN bus	Internal CAN bus
ABS	Disrupt ABS sensor reading by superimposing malicious magnetic field [111]	ABS sensor	Plant malicious device
	Spoof ABS sensor reading by inducing precise speed reading [111]	ABS sensor	Plant malicious device
Diagnostics (OBD...)			
Central Gateway	Force subnetwork message leak on the diagnostic bus by sniffing and spoofing [57]	OBD	OBD port
PassThru device	Obtain WiFi control offer PassThru device and access to reprogramming [23]	WiFi	Within WiFi range of PassThru device
	WiFi or wired malicious code injection [23]	WiFi	Within WiFi range of PassThru device
Security Access	Authenticate against various ECUs after obtaining keys and algorithm trough reverse engineering [83]	CAN bus	OBD port

Table 2.1: Reported attacks on various in-vehicle systems

2.3 Cryptographic components for checking integrity and authenticity

Cryptographic protocols are built by combining cryptographic primitives as fundamental building blocks. Cryptographic primitives are commonly divided in two main categories: *symmetric* and *asymmetric*. *Symmetric* primitives (e.g. hash functions, message authentication codes, symmetric-key encryption) are characterized by an even set of capabilities between the communicating parties. Usually, the symmetry mainly consists in the usage of identical keys both on the sender and receiver side. In the case of *asymmetric* primitives (e.g. asymmetric-key encryption, digital signatures) the capabilities of senders and receivers are uneven, a characteristic that is pointed out by the use of different keys: public and private.

Based on the security objectives that they provide, primitives can be further divided into several classes. As this thesis focuses on assuring authenticity we present details for primitives commonly employed for assuring this security objective, namely: hash functions, message authentication codes and digital signatures.

2.3.1 Hash functions

A *hash function* is a deterministic function that computes a small fixed length digest from a variable length input data. Hash functions must assure a set of security objectives. Having an input message x and the hash value for this input denoted as $H(x)$, the following properties must be satisfied: preimage resistance (for a given $H(x)$ it should be impossible to find the input message x),second preimage resistance (for a given pair $\{x1, H(x1)\}$ it should be impossible to find an input message $x2, x1 \neq x2$

such that $H(x1) = H(x2)$) and collision resistance (it should be impossible to find a pair $\{x1, x2\}, x1 \neq x2$ such that $H(x1) = H(x2)$).

Hash functions are used in various applications. Their main use is in verifying data integrity as checksum generators. Another major use of hash functions is in the construction of digital signature schemes or as the main building block of message authentication codes. They are also used in various computer applications for data indexing by building hash tables.

2.3.2 Message authentication codes

A message authentication code (MAC) is used to check the authenticity (which implies integrity) of data. MAC algorithms take a key and an arbitrary length message as inputs to generate a fixed length MAC. MACs are sometimes called message integrity codes (MIC) usually to alleviate confusion when using MAC as an acronym for media access control. However, in some cases, MICs are presented as algorithms that do not use secret keys, in this case they cannot provide authentication.

MACs are usually built based on other cryptographic primitives. Due to the often use of hash functions in their construction (e.g. HMAC [11]), MACs are sometimes called *keyed hash functions*. MAC algorithms such as CBC-MAC are built on block ciphers.

In terms of security objectives, MACs must resist forgery based on chosen-plaintext attacks. In other words, if an attacker has access to an oracle that knows the secret key and can generate MACs for any message chosen by the attacker, the attacker should not be able to guess MACs for messages other than those which have already been processed by the oracle.

In practice, MAC usage requires several steps. During an initialization phase, the secret keys must be shared between all communication participants in a secure manner. When a sender wishes to deploy a message, it computes the MAC over this message using the secret key and appends it to the sent message. Upon receiving a message, in order to verify message authenticity, the recipient must compute the MAC over the message and compare it to the received MAC.

The advantage of using MACs is that they are fast and built upon some of the simplest hash functions (or block cipers, see CBC-MAC) but on the downside they require sharing a secret key.

2.3.3 Digital signatures

Digital signatures are the building block for providing non-repudiation. Along with this property, they implicitly provide authenticity.

A digital signature scheme is composed of the following algorithms: an algorithm for key generation, that randomly chooses a private key and has the private and public keys as output, a signing algorithm that generates the signature based on the message and the private key and a signature verifying algorithm that checks the signature using

the message and the public key.

Signatures have the advantage that they require a single public key that can be used by any receiver to validate the authenticity of the message (not to mention that they also provide non-repudiation over the long run), however they are more computational intensive then simple MAC codes (ie. 100-1000 times)

3 Improving algorithm performance using chip-specific features

This chapter presents a performance evaluation of microcontrollers specific to the automotive industry along with improvements that can be achieved by using on-chip features such as parallelism and hardware cryptographic support. The details presented here cover results published in [85] and extended in [86]. Personal contributions to this work consists of the implementations, including parallelization as well as the performance evaluation and comparison of these implementations. To illustrate the performance of these embedded platforms we evaluate a series of hash functions. Our choice is motivated by the ubiquitous nature of hash functions in security mechanisms. Several practical scenarios in which hash functions are involved can be imagined, e.g., software validation, embedded communications, etc. In particular firmware updates in embedded platforms (which require cryptographic hash functions for the protection of intellectual property, data integrity or non-repudiation) can directly benefit from performance improvements. Notably, digital signatures are employed to ensure that only an authentic firmware is programmed on a certain embedded device [92]. Verifying signatures on a constrained embedded device can be a time consuming task especially as the size of the applications is continuously increasing [102]. The bigger the size of data to be flashed, the longer it will take to compute its hash value needed for signature verification, consequently deploying the framework on thousands of devices delays component delivery for days or even longer. Thus minimizing the overhead of security mechanisms on the production process is beneficial. At the very least, secure communication between embedded devices relies on secure gateways [125] that share secret keys and ultimately rely on MACs , i.e., keyed hashes. Obviously, many examples for the use of hash functions can be envisioned.

The reminder of this chapter is organized as follows. Section 3.2 presents formal approaches for parallelizing hash functions followed by specific implementation details for a set of selected hash functions in section 3.3. The performance analysis comes next in section 3.4.

3.1 Security on resource constrained devices

Assuring secure in-vehicle communication is not a straight-forward task. Besides the constraints given by the bus speed and payload, computational power and available memory also have to be taken into consideration. Microcontrollers that stand at the heart of ECUs are resource constraint devices in terms of speed and memory. Although more and more advanced microcontrollers are becoming available, the needs

of the industry are also continuously increasing making these constraints a persistent issue.

Implementing cryptography on resource constrained devices is a well and continuously investigated subject for which several solutions were successfully employed in practice. One category focuses on devising secure protocols which require little computational power and reduced variants of cryptographic functions. A good example in this area comes as a result of the intense research activity in sensor networks which produced solutions ranging from efficient protocol design to efficient cryptographic primitives [67]. Small scale variants of hash functions were also proposed for use in RFID environments which can be even more constrained than sensor networks [77]. However, collisions on these functions were already reported [114].

Another category of solutions are based on hardware implementations. Using ASIC or FPGA-based cryptographic hardware to perform the computation of required primitives increases performance along with the costs of production. Dedicated cryptographic coprocessors were developed to accelerate the execution of different primitives. Examples of such hardware implementations can be found in [96] and [115]. Some efforts were also made in enhancing the performance of general-purpose microcontrollers by extending their instruction set with application-specific instructions used in cryptographic algorithms [42]. Although they reach good performances, these hardware-based solutions are application dependent and require extra time to be spent on designing them in comparison to a software-based solution. A possible approach would be to use a standardized hardware security module such as the one developed for automotive platforms by the EVITA project [124]. As it is not clear if such solutions will be adopted by the automotive industry in the near future software, solutions based on microcontrollers that are already available on the market may be preferred in various contexts. Coming to the aid of developers, microcontroller manufacturers have also produced microcontrollers with integrated hardware support for well known cryptographic primitives [38]. These so-called cryptographic acceleration modules can implement the whole functionality of a certain primitive or only the segments that are more computationally intensive.

Microcontrollers are constantly evolving to handle the need for increased performance. Different manufacturers are already offering microcontrollers with on-chip general purpose coprocessors or even multi-core microcontrollers [104], [37]. This category of microcontrollers could be used to enhance cryptographic performance by using parallelism. One way to put this into practice is by running multiple instances of a function in parallel on each core to achieve high throughput. However, not all applications can rely on this kind of parallelism. Frequently, a single execution of a cryptographic primitive is needed at a certain time; so in order to attain speedups we have to search inside each individual algorithm for steps that can be parallelized. Some frequently used cryptographic algorithms were studied in this respect for the implementation of an FPGA-based crypto processor [19] and similar solutions are needed for multicore microcontrollers.

3.2 Using parallelism to improve algorithm performance

The performance of cryptographic algorithms can be improved by making platform specific code optimization. However, these optimizations only offer limited improvements and largely depend on the individual features of each microcontroller: instruction set, bit size, endianness, etc. Features that characterize each microcontroller lead to a performance bottleneck for sequential implementations. Therefore, where possible, techniques such as parallelism should be used in order to obtain further performance improvements.

Most cryptographic algorithms were not designed for parallel implementation and they mainly consist of chained steps, meaning that the output O of step S_{i-1} is used as an input for step S_i. However, some of the steps still can be executed in parallel. Even though in many cases only coarse-grained parallelism can be obtained, this is suitable for use on dual core microcontrollers.

In order to identify steps that can be executed in parallel the algorithm has to be divided in basic steps and input/output dependencies have to be determined. If the output of a step S_A is not needed in order to start execution of the next step S_B then the former can be run on a parallel processor. Another case for applying parallelism is when the output of a substep S_{Ai} is only needed by the corresponding S_{Bi}, i.e., $S_{Bi} = f(O_{Ai})$. Figure 3.1 shows how sequential steps in a) can be executed in parallel in b). We use these two observations in what follows to identify parallelizable steps which are specific for hash function. An illustration of how we apply these techniques on hash functions is depicted in Figure 3.2.b and a detailed explanation for this follows in the next section.

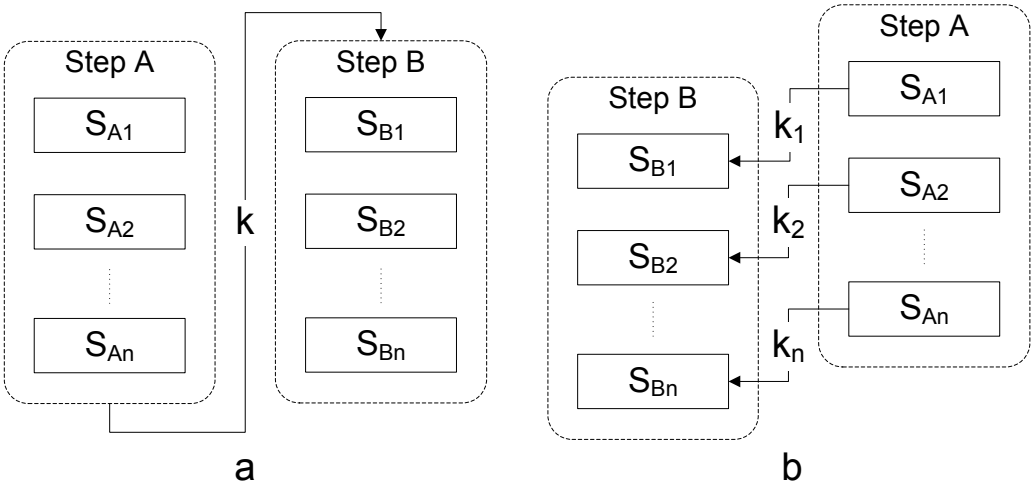


Figure 3.1: Transforming sequential execution (a) into parallel execution (b)

3.2.1 Parallelizing hash functions

Hash functions are probably the most commonly used cryptographic primitives. They generate a fixed length binary output for a variable length binary input. We focus on iterated hash functions as they are the most prevalent in this category. Although they accept arbitrary length input, hash functions work generally on fixed length blocks. Therefore the input message first has to be padded so that its length is a multiple of the hash block size. An initialization step follows and after it the input is passed through the compression function one block at a time. Figure 3.2.a shows the basic construction of an iterated hash function.

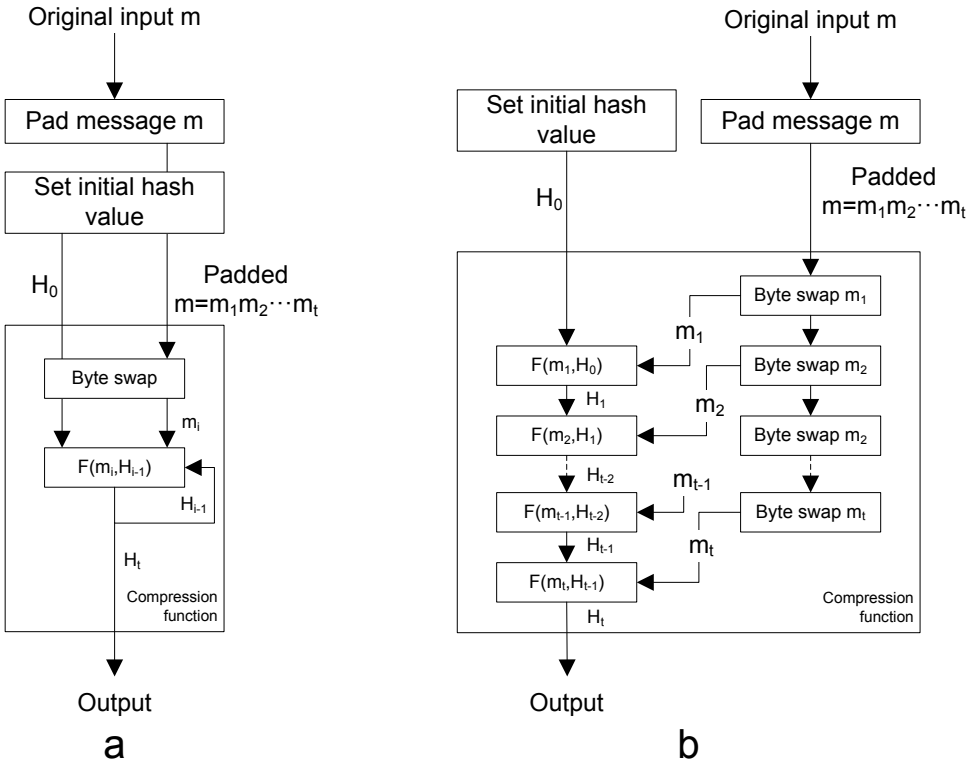


Figure 3.2: General construction of an iterated hash function. Sequential (a) and parallel (b)

Some parallelism is possible for this general hash function algorithm model. The padding step for example does not have to be executed sequentially. The only constraint here is that each padding byte has to be added to the input string before the compression function needs to start processing it. Performance gains obtained by parallelizing the padding step will obviously be negligible in the case of very long input messages but will make a difference when using short messages. A common substep of the compression function is one related to the endianness on which the algorithm

is based and the endianness of the used microcontroller. If the endianness differs, a byte swap to the correct endianness has to be made before processing the input and this can also be done in parallel with the main compression substeps. Further improvements could be made on the compression function depending on each design and we detail this in following sections for several chosen hash functions. Figure 3.2.b depicts the general parallelized construction for iterated hash functions.

Strongly related to hash functions are the message authentication codes (MAC), i.e., cryptographic primitives built for the purpose of providing message authentication. Given an input message of variable length and a key, these algorithms generate a fixed length output called MAC. Some of the most commonly used MAC constructions are based on other cryptographic primitives such as hash functions or block ciphers. Whether parallelism can be used in MAC algorithm implementations in general mainly depends on their individual construction and since they are quite differently built a rule of thumb cannot be stated. However, since most MAC algorithms are based on other cryptographic primitives, parallelism comes from the algorithms used as a building block where the rules given previously for hash functions apply. This also holds in the case of HMAC which is the most commonly used MAC construction and is built upon hash functions.

3.3 Implementation details

3.3.1 Parallelization within the SHA-3 candidates

Given the recent contest for designing the new hash standard, we included the SHA-3 candidates in our performance analysis. One evaluation criteria in the contest announced in 2007 by NIST for the next secure hash standard was the computational efficiency (referring to the speed of the algorithm). Therefore, the candidates were built so that they best fit this requirement. Some algorithms also exploited parallelism of different granularities suitable for hardware and/or software implementations [119]. We studied the candidates accepted for the final round of the SHA-3 contest in search for parallelism. Reference implementations available at [28] were used for benchmarking and as a starting point for our parallel implementations.

BLAKE [7] is based on the HAIFA iteration mode having a compression function which receives as inputs: a message block, a chain value, a salt and a counter representing the number of hashed bits so far and generates a new chain value. An inner round of the BLAKE compression function is a modified version of the ChaCha stream cipher and operates on a 4×4 state matrix of words. Independent operations are made on all four columns followed by operations on distinct disjoint diagonals. As the BLAKE specification states, this construction allows four-way parallelism so that operations on all four columns can be computed in parallel and identically for the diagonals. It is clear that for two core architectures (both cores equal in computational power) each core will take care of updating two columns and diagonals respectively. In our particular case, due to the increased speed of the coprocessor, we were able to

execute one column/diagonal update on the main CPU in parallel with updating three columns/diagonals on XGATE.

The developers of BLAKE, recently proposed a modified version called **BLAKE2** [8]. The changes included in BLAKE2 were aimed at bringing performance improvements without reducing security. BLAKE2 comes in two versions: BLAKE2b optimized for 64-bit architectures and BLAKE2s built for 8- to 32-bit platforms. We focus on the latter since it fits our target platforms. Changes present in BLAKE2 include: a reduced number of rounds and constants, speed-optimized rotations, minimized padding and endianness change. Due to the design similarities between the two versions the parallel implementation techniques previously presented for BLAKE also apply for BLAKE2.

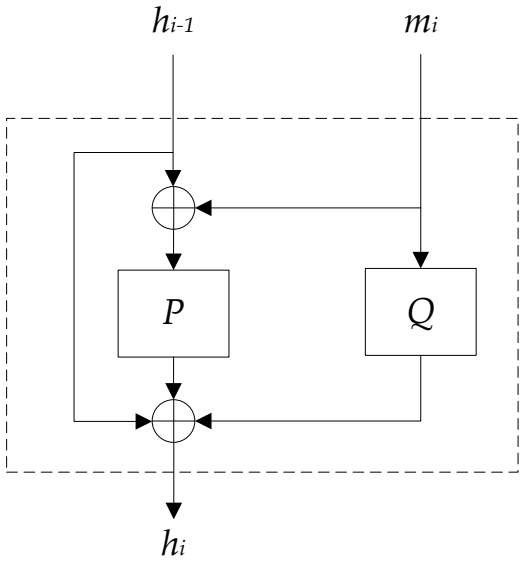


Figure 3.3: Grøstl compression function

Grøstl [40] is another iterated hash function which borrows some components from the AES construction. The Grøstl compression function is based on two fixed permutations. The two underlying permutations P and Q are combined as illustrated in Figure 3.3. These two permutations are completely independent of each other. Thus, implementing the compression function on two-way parallel architectures is quite straightforward especially as P and Q involve the same amount of computation. More in depth optimizations, by exploiting platform specific features, were done in [4].

JH [127] was built around a compression function F_8 which employs a large block cipher structure E_8. The first half of the $2m$ bit state is XORed with the current message block before feeding it to the E_8 permutation. The second half of the state computed by E_8 is also XORed with the current message block as shown in Figure 3.4. The E_8 construction uses 4×4 bit S-boxes, a linear 8-bit transformation L and a permutation P. An efficient implementation of JH utilizes the bit-slicing technique. In

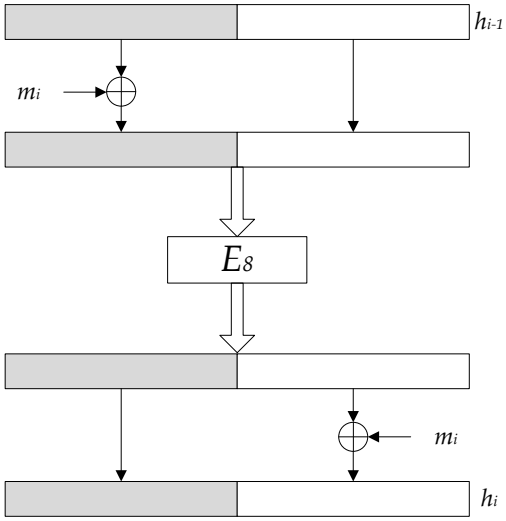


Figure 3.4: The F_8 compression function of JH

the bit-slice implementation, the 1024-bit input of E_8 is split in eight 128-bit words on which seven round functions (two S-boxes, one linear transformation L and four permutations ω) are applied. On a 128-bit processor, basic operations on each word would be done with one instruction while on a smaller bit size processor they will have to be constructed out of multiple instructions operating on smaller words. As the S-box and L functions can be executed independently on each corresponding bit of the eight words, parallelism can be used to speed-up the computation. The ω permutation swaps bits in a 128-bit word depending on the round number. This prevents bit level or nibble level parallelism for the full E_8 permutation. Therefore, we can only use nibble level parallelism in each round. Additionally, although not resulting in a large speed gain, the two XOR operation of the state and message block can also be made in parallel with other computations (E_8 round function for the first XOR and message truncation for the second).

Keccak [14] is defined as a family of sponge functions built on a set of seven permutations. Permutations are applied in two phases: one called absorbing phase, the second called squeezing phase, each based on the sponge construction. The $Keccak-f[1600]$ studied here consists of 24 rounds that operate on a state matrix of 5×5 lanes, where a lane is considered to be a 64-bit word. $Keccak-f[b]$ is a permutation over $\mathbb{Z}_2^b$. Each round consists of five steps designated by five Greek letters: θ, ρ, π, χ and ι. Some of these steps (ρ, π and χ) allow parallel execution as they are applied independently over distinct elements of the state matrix. More specifically, these steps can be independently applied overs sections of the state matrix by different cores. When using interleaving, as in the optimized implementation submitted by the Keccak designers (on which we based our implementation), the process of setting

bytes into interleaved words and vice versa can be also parallelized as it is applied on each word independently.

Skein [34] uses Threefish, a tweakable block cipher, running in Unique Block Iteration mode (UBI) to build its compression function. The core of Threefish employs three mathematical operations (one addition, one rotation by a constant and an XOR) to construct a simple mixing function called MIX on 128 bits. One round of Threefish in Skein-512 corresponds to the application of four MIX functions followed by a permutation of all 64-bit words in the state as depicted by Figure 3.5. Once every four rounds a subkey is injected in the current state. Skein-256 and Skein-1024 are similar, the only difference being that two mix functions are applied for a 256-bit state and eight for 1024. For parallel implementations each of the MIX functions in one round could be computed on a separate processor as they operate on independent 64-bit words of the state. As stated before, XGATE can handle several iterations of a certain block of code during a single execution of the same block on the S12X CPU. This allowed us to run 3 MIX functions on XGATE in parallel with one on S12X.

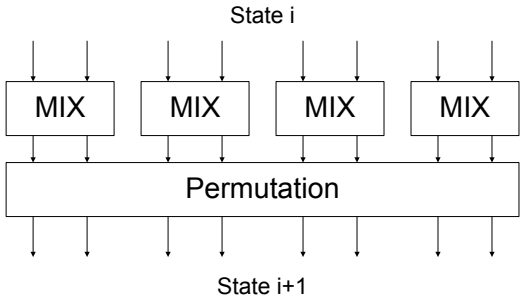


Figure 3.5: One basic round of Threefish-512

3.3.2 Parallelization within old hash standards

MD5 is a 128 bit output hash function developed by RSA Inc. as an improved version of MD4 [106]. It is still used in many applications although MD5 was proven to be insecure and collisions have been found since 2005 [122]. The MD5 algorithm is based on the general iterative hash function construction depicted in Figure 3.2. The compression function of MD5 operates on 512 bit message blocks divided in 16 4-byte words which have to be set in little-endian order before processing. As each step of the compression function uses the output of the previous step the construction is not suitable for parallelism. Thus, in general only the padding can be done in parallel. Also, on big-endian machines the words can be swapped to their little-endian form in parallel with the compression round.

SHA-1 [91] was deployed in various protocols and security standards since its release by NIST as a Federal Information Processing Standard. The construction of

SHA-1 is similar to the one of MD5 so the remarks on parallelism for MD5 also applies to SHA-1. The main differences are represented by the size of the message digest, the functions used by the compression function and initial hash value. Additionally in each compression round, the 64 bytes message block is expanded to a block of 320 bytes or 80 words. As the words in the message schedule obtained by expansion are processed one by one this expansion is independent of the compression process. Therefore the message expansion can be executed in parallel with the compression function.

SHA-2 consists of a set of four hash functions each having a message digest size denoted by its suffix: SHA-224, SHA-256, SHA-384 and SHA-512. The block size on which they operate is 512 bits for the first two and 1024 for the last two. As the algorithm structure of the SHA-2 family components is similar to the structure of SHA-1 (including the message expansion step) the same approach is to be used for adding parallelism.

3.3.3 Synthetical prediction of performance improvements

After identifying the steps that can be parallelized it is important to get an estimation on the improvement that could be obtained in order to decide whether or not to implement the changes. To be more accurate with our exposition, we first define what a *step* is. A step S_i corresponding to a logical part of an algorithm is a sequence of instructions that operate together as a group on a given input $I = (I_1 I_2 \cdots I_n)$ to generate an output $O = (O_1 O_2 \cdots O_m)$. Each step can be comprised of one or more *substeps* S_{ij}.

If we consider T_{CPU} as the time (measured in clock cycles) needed by the main CPU to execute a certain step then a coprocessor CPU will execute that same step in $s \cdot T_{CPU}$ clock cycles. Here $s > 0$ is a constant defining the coprocessor speed factor in relation to the main CPU. This element is influenced by a number of processor-specific features like: operation frequency, instruction set, type of memory used, etc. We denote the execution time of a step S_i on the main processor as T_i and on the coprocessor as $s \cdot T_i$. Let c_i be the maximum number of clock cycles that can be spent until the output of S_i is needed on the main processor and τ_i the time needed for exchanging data between the processors. We can assert that if $c_i \geq s \cdot T_i$ the entire step can be executed on the coprocessor, hence a speed gain of $T_i - \tau_i$. Otherwise, if $c_i < s \cdot T_i$ then only a fraction of c_i/s from the step was executed on the coprocessor while the rest can be executed either on the main CPU or further parallelized, etc. So we can summarize this gain as:

$$\begin{cases} T_i - \tau_i & \text{for} \quad c_i \geq s \cdot T_i \\ c_i/s - \tau_i & \text{for} \quad c_i < s \cdot T_i \end{cases} \tag{3.1}$$

However, in a two core environment such as S12X it is not efficient to swap the execution of a particular code sequence from the main CPU to the coprocessor. Therefore, in the case that a particular step once assigned to the coprocessor must be finalized there, we have to change this relation to:

$$\begin{cases} T_i - \tau_i & \text{for} \quad c_i \geq s \cdot T_i \\ T_i - (s \cdot T_i - c_i) - \tau_i & \text{for} \quad c_i < s \cdot T_i \end{cases} \tag{3.2}$$

Therefore, in the best case scenario the speed gain is T_i if the coprocessor can execute S_i in up to c_i cycles. Otherwise, if $s \cdot T_i > c_i$ the main processor will have to wait until the step is finished but still with a certain speed gain. It is clear that for very slow coprocessors the gain will be a negative number denoting that executing S_i on the coprocessor can't bring a speedup. Relation (3.2) can thus be written in a more compact form as:

$$min\,(T_i, c_i + T_i\,(1-s)) - \tau_i \tag{3.3}$$

Next by considering that each S_i is executed r_i times on the coprocessor, we can calculate the overall computational time reduction as:

$$\sum_{i=0,n} r_i\,(min\,(T_i, c_i + T_i\,(1-s)) - \tau_i) \tag{3.4}$$

As an example let us take BLAKE and try to estimate the speed gain for an input of length 0. Let s be 1/4 as XGATE is approximately 4 times faster than the main CPU. We identify four basic steps that can be parallelized on BLAKE: Byte Swapping (BS), Message Padding (MP), Round Operation (RO) on a column or a diagonal. On the main S12X CPU the computation time is:

- T_{MP} = 2213 cycles for message padding,
- T_{BS} = 2005 cycles for byte swap,
- T_{RO} = 1020 cycles for one round function.

The corresponding repetition times for each of these steps is: $r_{MP} = 1$ as the padding is only done once, $r_{BS} = 1$ because we have only one block and $r_{RO} = 14 \cdot 6$ because we have 14 rounds in which 3 columns and 3 diagonals are computed on XGATE. The execution time for all steps satisfies: $c_i \geq s \cdot T_i$. Therefore the actual speed gain can be computed as $T_{MP} + T_{BS} + 84 \cdot T_{RO}$ which gives 89898 cycles. This is close to the value presented in the experimental results, the difference coming from the τ_i which we did not consider in our calculus. We underline that this synthetic evaluation can provide only a rough measure of the speedup and only careful measurements can give the exact gain which depends on several other aspects such as memory, instruction set, etc.

3.4 Performance analysis and comparison

First, all cryptographic primitives described in the previous section were implemented on our automotive platforms in the sequential form. Then parallelism and hardware acceleration were used on platforms that allowed it. The block size of each

hash function implementation was selected to be 512 bits in order to allow comparative evaluation. $\lfloor Keccak[r = 1088, c = 512] \rfloor_{256}$, which we denote by Keccak-256, was used for Keccak. These implementations were analyzed in relation to two metrics: execution time and memory utilization. The next section presents the platforms employed before proceeding with the results obtained. Two of these platforms (S12X ad TriCore) are presented in more detail as they are used in other experimental setups throughout this thesis.

3.4.1 Employed platforms

3.4.1.1 The S12X platform

We used a SofTec Microsystems ZK-S12-B development board shipped with a Freescale 16-bit automotive microcontroller from the S12X family, MC9S12XDT512 [37]. All members of this family are equipped with a co-processor called XGATE. This co-processor was used to assess the performance that can be achieved for each function by employing parallelism.

The S12X microcontroller that was used has 512kBytes of flash memory and 20kBytes of RAM. Both FLASH and RAM memories are banked. Thus, a total of 8kBytes of RAM space can be used for continuous allocation while the rest of the RAM can be accessed in a 4kByte window. The data-sheet specifies that the maximum bus frequency that can be set using the PLL module is 40MHz. We configured the PLL for frequencies beyond this specified value and were able to go up to 80MHz without introducing any noticeable abnormal behavior.

The XGATE module has a built in 16-bit RISC core with instructions useful for data transfers, bit manipulations and basic arithmetic operations. The registers and instruction set of the XGATE differ from the ones available on the S12X core. The RISC core can access the internal memories and peripherals of the microcontroller without blocking them from the main S12X CPU. The S12X CPU always has priority when the two CPUs access the same resource at the same time. To assure data consistency, the access priority can be controlled by using the hardware semaphores available on the microcontroller. Figure 3.6 shows how the S12X core interacts with XGATE.

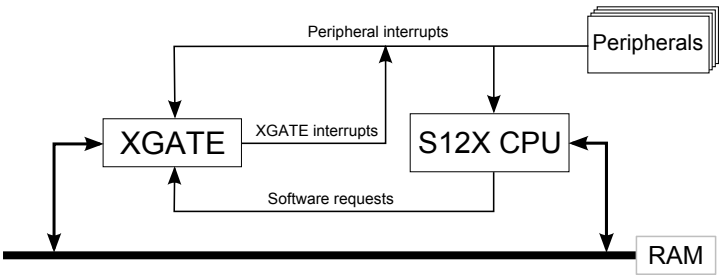


Figure 3.6: Interaction between the S12X core and XGATE

Interrupts can be routed to the XGATE module in order to decrease the interrupt load of the main S12X CPU. Additionally, up to 8 software triggered channels can be used by the S12X CPU to request software execution on XGATE.

In order to obtain the maximum XGATE CPU speed, the code can be executed from RAM. Because RAM is a volatile memory, XGATE code is being stored into the FLASH memory and copied into RAM after each reset. Having a better RAM access speed and a speed-optimized instruction set, a typical function can run up to 4.5 times faster on XGATE than on the S12X CPU [84]. Because it was designed for quick execution of small code requested by interrupts, the flash memory available for storing XGATE code is limited. For controllers in the S12XD family this limit is 30 kBytes.

The CAN module from the S12X chip can be programmed to function at bit-rates up to 1 Mbps. The limitation for the maximum achievable CAN speed on the S12 development board is given by the on-board low speed fault tolerant transceiver which can only run at speeds up to 125 kbps.

3.4.1.2 The K60 platform

To evaluate the performance improvements of a crypto accelerator the Kinetis K60 microcontroller was used. This Freescale device is built around a 32-bit ARM Cortex-M4 core and has 512 Kbytes of FLASH and 128 Kbytes of RAM allowing for a maximum working frequency of 100 MHz. The K60 features on-chip hardware support for CRC and random number generation as well as a cryptographic acceleration unit for speeding up execution for a set of well known ciphers and hash functions: DES, 3DES, AES, MD5, SHA-1 and SHA-256. In terms of connectivity, the K60 is capable of Ethernet and high-speed CAN communication.

3.4.1.3 The TriCore platform

We also used Infineon TriCore microcontrollers from the AUDO (AUtomotive unifieD processOr) family. All members of this family are 32 bit microcontrollers built for computational performance. Experiments were done using two members of this family: TC1797 and TC1782. The TC1797 microcontroller [62] is a member of the AUDO Future family and can work at frequencies up to 180MHz, with a program flash memory of 4MB and a 64kB data flash along with 176kB of SRAM. The TC1782 [61] is a mid range automotive microcontroller and a member of the newer AUDO MAX family (but it is still based on the same 1.3.1 core version as TC1797). TC1782 has the same operating frequency and SRAM size as TC1797 while the program flash memory is reduced to 2,5MB and the data flash is double in size to hold 128kB. Due to the similar architectures, the computational performance of these two microcontrollers is almost identical. Thus, in this section we only present results obtained from the TC1797 whenever referring to the TriCore platform.

Both TriCore controllers have CAN and FlexRay modules enabling communication on these special bus types. The development boards are equipped with high speed CAN transceivers that can be configured to run at a maximum speed of 1 Mbps.

3.4.2 Execution speed

As a first measure of performance we chose to evaluate the execution speed of the algorithms in their normal versus improved implementations. The execution time was measured as the number of clock cycles needed to hash input messages of different lengths. The range of input lengths were selected to illustrate the performance of each function for both small and big messages. A finer grained evaluation was made for input sizes up to 256 bytes to reflect the typical step shape of the performance curve that depends on the way that each hash function handles message blocks. For each implementation we show a graph containing hash function performance curves for comparison. Next the speed is presented as cycles per byte in a tabular form along with the performance gains for a selection of input lengths. An estimation for very long messages, synthetically computed [29], is also added. Finally we compare our results with data from related work.

S12X. Figures 3.7 and 3.8 show performance curves for the S12X sequential and parallel implementations. Values on the X axis represent message input lengths and the Y axis corresponds to the number of clock cycles spent to hash the message. In the case of the S12X sequential implementation, the superimposed curves show that BLAKE is the best SHA-3 performer followed by Grøstl and Skein while Keccak and JH are positioned last following SHA-256. The ranking is slightly changed in the parallel implementation with Keccak and JH following BLAKE and Grøstl. SHA-256 is next while Skein is the last ranked. In both cases BLAKE2s proves to be faster than the SHA-3 candidates and comparable in speed with MD5 for the parallel implementation. Table 3.1 shows cycle/byte representations of our experimental data for selected input values and summarizes the improvements obtained for parallel implementations of each function. The best improvement, around 73%, was obtained for BLAKE and JH, followed by BLAKE2 and Keccak. Grøstl comes next with a gain comparable with the ones obtained for SHA-1 and SHA-256 for small inputs, but it increases for larger

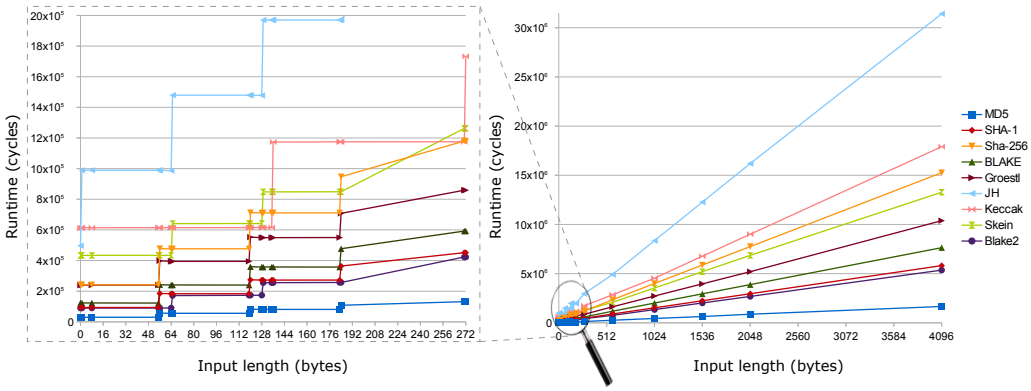


Figure 3.7: Runtime of sequential implementation (S12X)

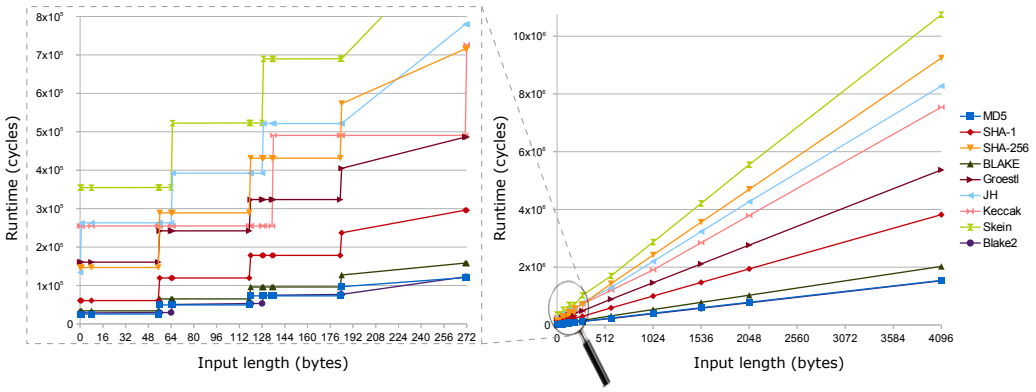


Figure 3.8: Runtime of parallel implementation (S12X)

inputs. From the SHA-3 candidates Skein had the smallest speedup. MD5 comes last in terms of improvement as the compression function does not offer significant means for applying parallelism.

For all the hash functions we wanted to see if the effect of the padding step over the total runtime can be alleviated through parallel implementations. Hence the input sizes were selected so that best/worst cases (minimum/maximum number of padding bytes needed) are covered in the tests. Table 3.2 shows the execution time for the chosen inputs. It can be easily observed that the parallel implementations spend the same amount of time for all inputs with the same block count after padding since the padding step is not done sequentially anymore. As the padding rules differ from one hash function to another, the block count changes at different input sizes. For example in JH each message is padded so that its length is a multiple of 512 bits adding at least 512 bits (when the message is already a multiple of 512) and at most 1023 bits are added to the message. However, some exceptions from this rule exist in the case of BLAKE which contains an additional step in the compression function executed if the last block only contains padding bytes, hence the results obtained.

Kinetis K60. All the hash functions analyzed on the S12X platform were also evaluated on the Kinetis K60 microcontroller. We investigated the possibility of enhancing the performance of hash functions included in this study with the K60 Memory Mapped Cryptographic Acceleration Unit (MMCAU). Each MMCAU command is called by write operations at locations in a specially designated memory space. Up to three commands can be written with a write operation (to reduce the time spent for command calling), commands will, however, be executed sequentially. Results are obtained by similar read operations. In the case of MD5 and previous SHA standards the use of MMCAU is straight forward since it provides specialized commands for each of their round function operations. To identify possible uses of the MMCAU for the other hash functions we have to analyze their algorithms.

As previously described, Grøstl is one of the SHA-3 contest finalists which ben-

	Input size (bytes)	0	8	64	128	576	1024	1536	4096	long input
MD5	Sequential (cycles/byte)	29837	3748.4	865.2	631.5	449.7	427	417.3	405.1	384.8
	Parallel (cycles/byte)	25672	3209	771.5	571	415	395.5	387.1	376.7	370.4
	Improvement (%)	**13.96**	**14.39**	**10.82**	**9.58**	**7.72**	**7.38**	**7.22**	**7.01**	**3.75**
SHA-1	Sequential (cycles/byte)	93826	11747	2863.2	2129.6	1559	1487.7	1457.1	1418.9	1396
	Parallel (cycles/byte)	60791	7598.9	1867.8	1392.9	1023.4	977.3	957.5	932.7	917.9
	Improvement (%)	**35.21**	**35.31**	**34.77**	**34.60**	**34.35**	**34.30**	**34.29**	**34.26**	**34.35**
SHA-256	Sequential (cycles/byte)	241691	30230.1	7443	5554.2	4085.1	3901.5	3822.8	3724.4	3665.4
	Parallel (cycles/byte)	146812	18351.5	4515.3	3368.3	2476.2	2364.7	2316.9	2257.2	2221.3
	Improvement (%)	**39.26**	**39.29**	**39.34**	**39.36**	**39.38**	**39.39**	**39.39**	**39.39**	**39.40**
BLAKE-32	Sequential (cycles/byte)	122755	15375.1	3752.7	2793.6	2047.7	1954.5	1914.5	1865.2	1835.9
	Parallel (cycles/byte)	33780	4241.4	1013.9	750	544.7	519	508	494.3	486.1
	Improvement (%)	**72.48**	**72.41**	**72.98**	**73.16**	**73.40**	**73.44**	**73.46**	**73.50**	**73.53**
Grøstl-256	Sequential (cycles/byte)	239973	30001.8	6168.3	4293.5	2835.3	2653.0	2574.9	2535.8	2535.8
	Parallel (cycles/byte)	160765	20095.6	3783.2	2527.2	1550.3	1428.2	1375.9	1310.5	1271.3
	Improvement (%)	**33.01**	**33.02**	**38.67**	**41.14**	**45.32**	**46.17**	**46.57**	**48.32**	**49.87**
JH-256	Sequential (cycles/byte)	496546	123665.9	15422.8	11543.6	8526.4	8149.2	7987.6	7669.6	7432.4
	Parallel (cycles/byte)	133826	32874	4109.3	3063.7	2250.6	2148.9	2105.4	2020.2	1956.8
	Improvement (%)	**73.05**	**73.42**	**73.36**	**73.46**	**73.61**	**73.63**	**73.64**	**73.66**	**73.67**
Keccak-256	Sequential (cycles/byte)	613917	76782.6	9605.7	4807.4	4954	4426.6	4408.4	4372.1	4344.9
	Parallel (cycles/byte)	254898	31862.3	3982.8	1991.4	2079.6	1860.4	1854.2	1841.8	1832.5
	Improvement (%)	**58.48**	**58.50**	**58.54**	**58.58**	**58.02**	**57.97**	**57.94**	**57.87**	**57.82**
Skein-512-256	Sequential (cycles/byte)	433565	54215.8	6784	5012.8	3631.2	3458.5	3384.5	3240	3132.5
	Parallel (cycles/byte)	354395	44319.5	5547	4085.1	2946.4	2801.4	2740	2622.8	2535.7
	Improvement (%)	**18.26**	**18.25**	**18.23**	**18.51**	**18.86**	**19.00**	**19.04**	**19.05**	**19.05**
BLAKE2s	Sequential (cycles/byte)	89636	11230.4	1411.7	1357.8	1317.7	1312.6	1310.5	1307.8	1306.2
	Parallel (cycles/byte)	28425	3588.8	465.3	417.35	417.3	377.4	375.4	373.1	371.6
	Improvement (%)	**68.29**	**68.04**	**67.04**	**68.27**	**71.03**	**71.25**	**71.35**	**71.47**	**71.55**

Table 3.1: Speed (cycles/byte) and improvements on several input sizes (S12X)

Input size (bytes)		0	55	56	64	65	119	120	128	129	135	136
MD5	Sequential	29837	29987	57239	55371	55445	55445	82697	80829	80903	80903	80903
	Parallel	25672	25672	49378	49378	49378	49378	73084	73084	73084	73084	73084
SHA-1	Sequential	93826	93976	185111	183243	183317	183317	274452	272584	272658	272658	272658
	Parallel	60791	60791	119537	119537	119537	119537	178283	178283	178283	178283	178283
SHA-256	Sequential	241691	241841	478219	476351	476425	476425	712803	710935	711009	711009	711009
	Parallel	146812	146812	288978	288978	288978	288978	431144	431144	431144	431144	431144
BLAKE-32	Sequential	122755	122680	242263	240172	240414	240093	359676	357585	357827	357827	357827
	Parallel	33780	33931	65038	64887	65038	65038	96145	95994	96145	96145	96145
Grøstl-256	Sequential	239973	240248	398361	394769	394775	395044	553158	549566	549571	549601	549556
	Parallel	160765	160765	242124	242124	242124	242124	323483	323483	323483	323483	323483
JH-256	Sequential	496546	989139	989135	987062	1479871	1479655	1479651	1477578	1970387	1970363	1970359
	Parallel	133826	262992	262992	262992	392158	392158	392158	392158	521324	521324	521324
Keccak-256	Sequential	613917	614684	614693	614765	614774	615260	615269	615341	615350	615404	1173876
	Parallel	254898	254898	254898	254898	254898	254898	254898	254898	254898	254898	490639
Skein-512-256	Sequential	433565	434150	434159	434177	641129	641616	641624	641643	848262	848316	848325
	Parallel	354395	354395	354395	354395	522375	522375	522375	522375	689475	689475	689475
BLAKE2s	Sequential	89636	90266	90275	90347	171529	173473	173509	173797	255125	255341	255377
	Parallel	28425	29604	29626	29777	50519	53000	53044	53411	74299	74573	74617

Table 3.2: Speed (cycles/byte) and improvements for input sizes selected to illustrate effects of parallelizing the padding step (S12X)

efits from the use of AES transformations. The P and Q permutations that are at the core of the Grøstl round function consist of several rounds in which four AES-like transformations are applied: AddRoundConstant, SubBytes, ShiftBytes and MixBytes. SubBytes applies the AES S-box to each byte in the state matrix, while ShiftBytes and MixBytes are variations of the corresponding ShiftRows and MixColumns steps in AES. Therefore, in the case of K60 we can use the MMCAU AES SubBytes command. Our MMCAU implementation based on the Grøstl reference code only showed an improvement of 3.4%. The 32-bit optimized implementation submitted by the Grøstl team which is based on look-up tables has a much better performance (60% faster than our MMCAU-based implementation) at the cost of a greatly increased code size (our implementation needs 85% less code memory).

The rest of the studied hash functions were not built using building blocks of previous primitives. The MMCAU commands could not be used to improve their performance.

Figure 3.9 offers a graphical representation of these results. All K60 implementations of SHA-3 finalists are slower than old SHA standards. Keccak, Skein and BLAKE closely follow SHA-256 while Grøstl and JH rank last at a significant distance. Here BLAKE2 proves to be slightly faster than SHA-256. Table 3.3 backs up the plots with hashing speeds represented as cycles per byte. MD5, SHA-1 and SHA-256 were also implemented using HW support and Table 3.3 also provides performance data for these implementation. As mentioned before, the fastest Grøstl implementation could not be outranked by using HW support on the K60 platform. In what concerns the other hash functions, they could also not benefit from speed-ups using the K60 MMCAU. However, for the three presented MMCAU implementations adding hardware acceleration resulted in smaller speedups for smaller and faster hash functions such as MD5 and SHA-1 than for the more time-consuming ones like SHA-256.

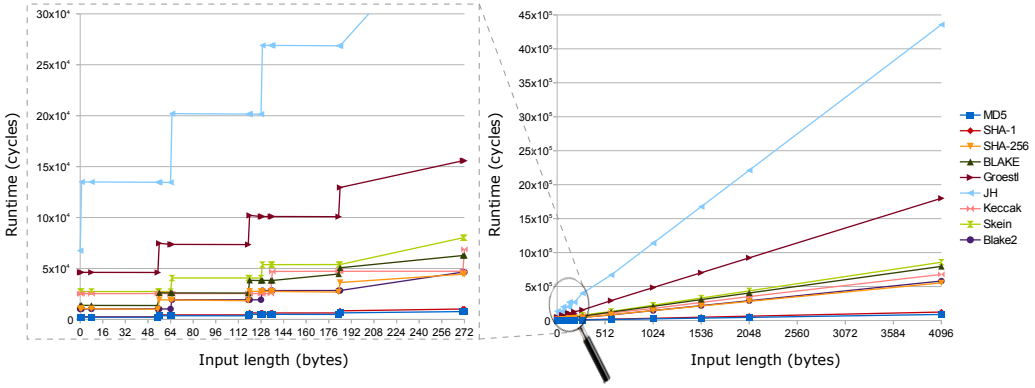


Figure 3.9: Runtime without HW support (K60)

TriCore. On the TriCore platform, the performance of hash functions was investigated for the standard sequential implementations. Execution speeds for various input

Input size (bytes)		0	8	64	128	576	1024	1536	4096	long input
MD5	No HW support (cycles/byte)	2120	278.8	55	38.1	25	23.3	22.7	21.8	21.2
	With HW support (cycles/byte)	1545	183.3	36.1	24	14.6	13.4	12.9	12.3	11.9
	Improvement (%)	**27.12**	**34.26**	**35.93**	**36.99**	**41.61**	**42.43**	**42.96**	**43.50**	**43.91**
SHA-1	No HW support (cycles/byte)	2476	318.1	69	49.4	34.2	32.3	31.5	30.5	29.8
	With HW support (cycles/byte)	2138	255.3	53.2	36.5	23.5	21.9	21.2	20.3	19.8
	Improvement (%)	**13.65**	**19.76**	**22.82**	**26.08**	**31.19**	**32.63**	**31.68**	**33.23**	**33.61**
SHA-256	No HW support (cycles/byte)	10561	1316.1	297.6	215.1	150.9	142.9	139.4	135.1	132.6
	With HW support (cycles/byte)	2521	303.1	64.7	45.0	29.7	27.8	27	25.9	25.3
	Improvement (%)	**76.13**	**76.97**	**78.25**	**79.06**	**80.31**	**80.55**	**80.66**	**80.80**	**80.89**
BLAKE-32	No HW support (cycles/byte)	13714	1727.1	405.6	298.4	214.9	204.5	200	194.5	191.1
Grøstl-256	No HW support (cycles/byte)	46245	5777.6	1150.6	789.3	508.3	473.2	458.2	439.3	428.1
JH-256	No HW support (cycles/byte)	67474	16869.4	2101.2	1574.1	1164.1	1112.9	1090.9	1063.4	1047
Keccak-256	No HW support (cycles/byte)	25232	3169.1	396.4	198.3	195.5	173.8	172.6	165.7	159.5
Skein-512-256	No HW support (cycles/byte)	27325	3421.3	427.1	317.2	230.8	220	215.3	209.5	206.1
BLAKE2s	No HW support (cycles/byte)	10218	1286	160.9	150.8	143.7	142.8	142.4	142	141.7

Table 3.3: Speed (cycles/byte) and improvements for different input sizes (K60)

Input size (bytes)	0	8	64	128	576	1024	1536	4096	long input
MD5 (cycles/byte)	2133	264.8	46.8	30.4	17.6	16	15.3	14.4	13.9
SHA-1 (cycles/byte)	3108	387.5	72.6	48.5	29.8	27.4	29.9	25.1	24.4
SHA-256 (cycles/byte)	6615	814.1	181.1	130.3	90.8	85.9	83.8	81.1	79.5
BLAKE-32 (cycles/byte)	5360	645.1	136.8	96.8	65.7	61.8	60.1	58	56.7
Grøstl-256 (cycles/byte)	42660	5013	1010.3	690.2	451.3	419.6	408.8	395.2	386.7
JH-256 (cycles/byte)	18738	4308.8	537.5	397.4	288.8	274.9	269.1	261.7	257.2
Keccak-256 (cycles/byte)	17719	2214.2	276.8	138.4	113.2	97.6	95.3	89.4	84.8
Skein-512-256 (cycles/byte)	33030	4126.5	515.8	380	273.6	260.3	254.7	247.5	243.3
BLAKE2s (cycles/byte)	5034	632.9	79.1	60.3	46	44.2	43.5	42.5	41.9

Table 3.4: Speed (cycles/byte) and improvements for different input sizes (TriCore)

sizes are summarized in Table 3.4 and illustrated in Figure 3.10. As depicted by these results, on the TriCore platform, MD5 and SHA-1 are the best performers followed by BLAKE2 and BLAKE. SHA-2 and Keccak come next with similar performances, while Skein, JH and Grøstl come last.

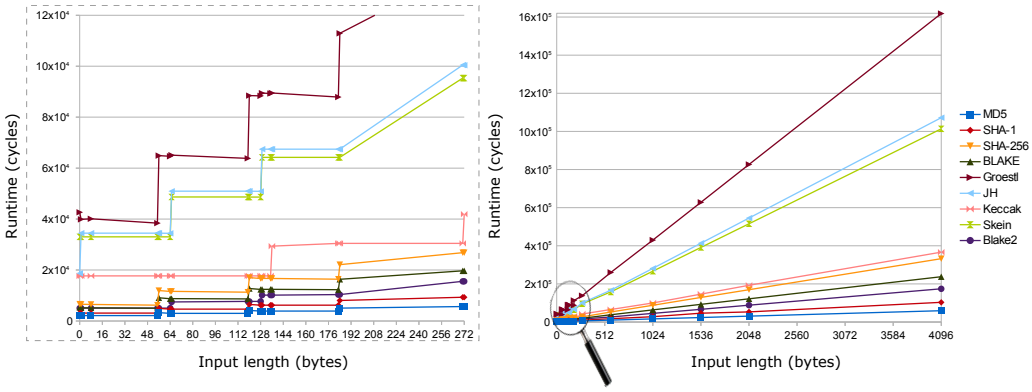


Figure 3.10: Runtime of sequential implementation (TriCore)

Comparison with related work. To create a more accurate image on the performance of the SHA-3 candidates on low performance platforms we put our results alongside related results obtained on similar platforms. We chose the XBX benchmarks [123] as they are based on a variety of 8-, 16- and 32 bit platforms close in performance to ours. Figures 3.11 and 3.12 show performances measured on differ-

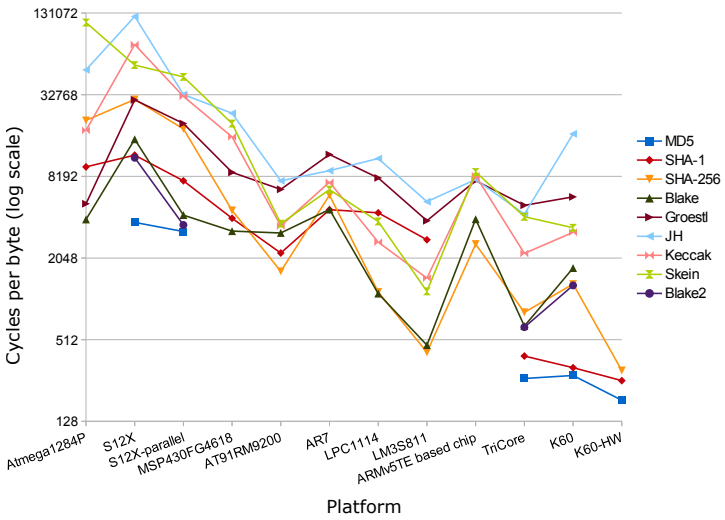


Figure 3.11: Performance comparison on different platforms for 8 byte inputs

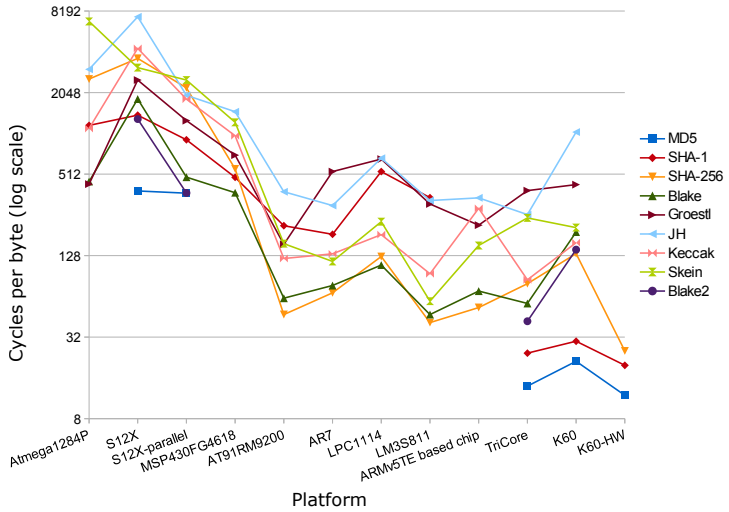


Figure 3.12: Performance comparison on different platforms for very long inputs

ent platforms for an 8 byte and very long inputs respectively. The X axis holds the platforms added for comparison ordered by microcontroller register width. The first (from left to right) is an 8-bit microcontroller followed by our S12X platform and one other 16-bit architecture. The rest of the platforms are 32-bit. The Y axes represents the hashing speed in cycles per bytes on a logarithmic scale, thus the lowest points on the graph indicate the fastest algorithms on each platform.

As the graphs show, BLAKE remains the best performer from the five SHA-3 finalists. Grøstl is the second in line for the 8- and 16-bit platforms but occupies one of the last places for the 32-bit architectures. Keccak and Skein have comparable performances across all platforms, the only exceptions being in the case of the 8-bit microcontroller and that of the TriCore platform. Finally, JH comes last or second-to-last in the majority of cases. SHA-2 shows similar performances to BLAKE for 32-bit platforms but it is surpassed by some of the SHA-3 candidates on the other 8 and 16-bit platforms.

3.4.3 Memory utilization

Table 3.5 illustrates the code size of our implementations on both presented platforms. From the SHA-3 finalists, BLAKE and JH show the lowest memory consumption in S12X sequential implementation followed by Skein, Grøstl and Keccak. For parallel execution BLAKE remains the best performer as it requires the smallest amount of memory. Where it was permitted, without compromising performance, we took advantage of the increased speed of XGATE to optimize code not for speed but for size. For example, in the case of MD5, SHA-1, SHA-256 and Skein (since XGATE finishes parallel computations faster than the main S12X core anyway and then it will have

Function	Code size S12X		Code size K60		Code size TriCore
	Sequential	Parallel	No HW support	HW support	Sequential
MD5	4243	3701	2634	1858	3476
SHA-1	8323	4144	4454	3362	6040
SHA-256	17145	5437	10580	1010	1294
BLAKE-32	5073	6685	11090	-	4107
Grøstl-256	15434	27414	25372	-	13012
JH-256	5912	17675	9432	-	5766
Keccak-256	22205	27132	3490	-	14398
Skein-512-256	13726	11079	19850	-	50158
Blake2s	6457	6425	10678	-	8966

Table 3.5: Memory consumption (bytes)

to wait) we gave up loop unrolling resulting in a smaller overall memory consumption than in the sequential implementation. Otherwise, code size for parallel implementation is generally larger, primarily due to the differences between the S12X main core and XGATE instruction set. If we take JH as example, the exact same code for the E8 compression function needs approximately three times more space on XGATE than on the main S12X core due to the reduced instruction set.

On the Kinetis K60 side, Keccak has the smallest code size followed by JH and BLAKE while Skein and Grøstl come last at almost double the size of BLAKE. When using hardware acceleration the code size also decreases. Similarly to the case of speedups, SHA-256 has the biggest gain in regard to memory consumption.

When using the TriCore platform, the SHA-256 implementation has the lowest memory consumption. MD5, BLAKE, JH and SHA-1 come next followed at a distance by Grøstl and Keccak. Skein is last with the highest memory consumption from the set of studied functions.

3.4.4 Discussion

Experimental results derived from our implementations showed that, when using parallelism, the execution speed can be improved with factors in the range of 10 - 70% depending on the algorithm and coprocessor used. In particular, parallel implementations of the SHA-3 finalists can run 18% to 73% faster. These values hold for a scenario on which the XGATE coprocessor is up to 4.5 times faster than the CPU (the actual speed of XGATE depends on various parameters: memory type used for storing code and data, instructions used, etc.). In most cases this improvement comes at the cost of a tradeoff in what concerns the used memory. However for some algorithms and on coprocessors that run faster than the main CPU, speedups can be obtained along with a decrease in code utilization when applying code size optimizations. When using hardware accelerated implementations speedups come along with a reduction of the code size. For the K60 implementation of SHA-256 we were able to obtain an improvement of up to 80%.

As expected, MD5 stands out as the best performer in terms of fast execution as

well as low memory requirements. Although proven to be insecure MD5 is still widely used in various applications. Due to the real-time nature of automotive systems we consider MD5 as being sufficiently secure. Moreover, due to its increased performance we subsequently use it in all our constructions to achieve a bottom line for protocol performance.

4 Application layer authentication

To withstand threats, in-vehicle networks, more specifically, those based on CAN buses can be protected by implementing security at the application level as the communication protocol has no intrinsic security mechanism. This chapter looks at solutions for achieving authenticated communication on CAN. First we go through the related work in this area and present existing solutions for broadcast authentication in section 4.1. Next, three approaches for assuring higher layer authentication are described and evaluated. The first one is based on the TESLA protocol and presented in section 4.2 which holds results published in [44, 46, 47]. The second approach employs one-time signatures based on lightweight cryptography and is presented in section 4.3. This section holds results published in [45] and [48]. The third protocol, called LiBrA, uses keys which are split between groups of nodes and lightweight cryptographic primitives to achieve authentication. Section 4.4 contains details on the LiBrA protocol and its performance based on results published in [49]. Personal contributions consist of all implementations and evaluation of protocol performance on chosen platforms.

4.1 Related work

4.1.1 Related work on secure broadcast protocols

Although digital signatures provide an elegant method for signing broadcast data, they are not the solution in our context because of both the computational and communication overhead. As messages are short in CAN networks, usually fitting in the 64 bits of data carried by one CAN frame, using a public-key primitive such as the RSA requires thousands of bits and causes a significant overhead. Elliptic curves can significantly reduce the size of the messages, but still the computational overhead is too much to assure small authentication delays. While the computational overhead can be alleviated by using dedicated circuits, such as ASICs and FPGAs, this would increase the cost of components, an issue that is largely avoided by manufacturers. One alternative to digital signatures such as RSA, or ECDSA is the use of one-time signatures which were initially proposed by Merkle in [82]. Although they are frequently mentioned in the literature as a cheaper alternative to conventional signatures, they are quite unused in practice, mostly because of their one-time nature. Using Merkle trees makes them viable for multiple uses, but it requires sending an entire path of a tree, and generating, potentially storing all this tree on the signer side, which requires even more resources. There is good literature available on the subject but this line of

research appears to have a reduced practical impact as one-time signatures are not so common in practice.

In contrast, symmetric primitives were efficiently employed for authentication in constrained environments such as sensor networks [75, 76, 100]. Due to the broadcast nature of CAN, protocols similar to the well known TESLA protocol [101], [99] can be used in this context as well. Indeed, some of the constraints are similar. For example, computational power is also low and, although high speed microcontrollers are also available on the market, low speed microcontrollers are preferred to reduce costs.

While TESLA like protocols introduce delays that could be unsuitable for all real-time CAN based applications, there is a broad area of applications where they could be tolerated in exchange for security. In particular, delays in the order of milliseconds, or even below as proved to be achievable by our proof-of-concept implementation, are suitable for a broad range of in-vehicle control tasks. A version of the scheme that achieves immediate authentication is in fact available from Perrig et al. in [99] but this scheme addresses the case in which the MAC on the message is sent before the key disclosure while the message itself is sent afterwards (allowing to authenticate the message when it is received). Here by immediate authentication we mean that, as soon as a principal knows the value of the message, he can broadcast it and its authenticity can be checked by receivers as soon as the authentication tag is received.

There is an extensive bibliography related to the TESLA protocol. Its history can be traced back to Lamport's scheme which uses one-way chains to authenticate users over an insecure network [71]. The work of Bergadano et al. [13] proposes several variants of one-way chain based protocols, with or without time synchronization. Previous work which inspired this family of protocols is the Guy Fawkes protocol from [1]. The context which is more related to our setting here is that of the application of such protocols in sensor networks. In particular, several trade-offs for sensor networks were studied by Liu and Ning in [75], [76] and several variants of the protocols are presented by Perrig et al. as well in [101], [99].

Also several papers addressed hybrid versions in which asymmetric primitives are mixed with key-chains in order to obtain trade-offs [12], [21], [5]. However, these variants have a bigger communication or computational overhead and do not appear to be appropriate for our application setting.

4.1.2 Related work on authentication in controller area networks

While most of previous research in the automotive area was focused on vehicle to vehicle and vehicle to infrastructure communication there seem to be only a few results for assuring security on communication buses inside vehicles. Most of the approaches to in-vehicle security advocate the use of secure gateways between different ECUs (Electronic Control Unit) or subnetworks [9], [125] and rely on basic building blocks from cryptography (encryptions, signatures, etc.). However, none of these approaches is meant specifically for assuring broadcast authentication on CAN which is still the

most common communication bus in automotives.

In this respect, two main results in assuring CAN security can be found so far, one of them proposes a new paradigm which closely follows CAN specifications [120] while the other introduces a validity voting scheme in [116] and [117].

CANAuth. CANAuth [120] is a protocol that has the merit to follow in great detail the specifications of CAN, its security is specifically designed to meet the requirements of the CAN bus. In particular, CANAuth is not intended to achieve source authentication as the authentication is binded to the message IDs and messages may originate from different sources which will be impossible to trace. This fits the specification of CAN which has a message oriented communication. However, a first issue is that the number of CAN IDs is quite high, in the order of hundreds (11 bits) or even millions in the case of extended frames (29 bits) and storing a key for each possible ID does not seem to be so practical. For this purpose, in [120] a clever solution is imagined: the keys are linked to multiple ID codes using masks, which greatly reduces the required amount of keys. But still, this leads to some security concerns, which we discuss next. Traditionally, keys are associated to entities to ensure that they are not impersonated by adversaries, but the effect of associating keys to messages is less obvious. For example, any external tool (assume On-Board Diagnostics (OBD) tools which are wide spread) that is produced by external third parties will have to embed the keys associated for each ID that it sends over or even just listens to on CAN, provided it needs to be able to authenticate those IDs. It is thus unclear which keys can be shared with different manufacturers and how or what the security outcomes of this are. Obviously, if a third party device, even an innocuous one such as passive receiver, is easy to compromise then all the IDs which it was allowed to authenticate are equally compromised.

Voting. Szilagyi and Koopman introduce a validity voting scheme in [116] and [117]. The scheme is intended for generic time-triggered communication such as TT-CAN, FlexRay, etc. The core part of the protocol relies on the classical paradigm of sharing keys between each sender and receiver then authenticating packets on a one MAC per receiver basis. Further, to make it feasible to embed the MACs in a single frame, the tags are truncated and concatenated (e.g., 3 MACs each of 8 bits are fitted at the end of a single frame). The communication is time triggered, each receiver releasing his message and his vote on previous messages in fixed time slots. Both the new message and his vote, along with all previously received messages, are authenticated under the same array of MACs to other receivers. The scheme appears to be a trade-off between computational time, authentication delays and bandwidth in order to fit the authentication bits in one frame. Indeed, if the frame would be larger, and the sender could fit more MAC bits in each frame, then authentication could be done at once within a single frame without needing to wait for the votes of the other nodes. This would improve both on delays (as nodes will not need to wait for the vote of other nodes) and computational power since, indeed, the nodes that subsequently vote are re-authenticating messages that were previously authenticated with a small amount of bits. The procedure leads to a drawback as stated in [117]: for frames that are

lost, the receive history of the nodes does not match and authentication will fail for these frames. As suggested in [117] this can be fixed by adding additional bits for lost packets, but sufficient votes from other nodes would still be required to deem the frame authentic.

4.2 TESLA-based CAN authentication

As a first approach we look at the well known TESLA broadcast authentication protocol. The main argument for choosing a TESLA like protocol in our research is that there is no better solution to perform broadcast authentication without secret shared keys or public key primitives. Also, there is no result so far, to best of our knowledge, that points out clear technical limits on using TESLA like protocols on CAN networks. Thus, we provide clear experimental results on two automotive microcontrollers located somewhat on the extremes of computational power in terms of memory and bus speed: a Freescale S12 equipped with an XGATE coprocessor and an Infineon TriCore.

The results presented here are relevant as the authentication delay is critical for control scenarios. This is different to the usual sensor-network scenario where TESLA like protocols are frequently used because in sensor networks other constraints are more prevalent. For example, energy consumption is a critical issue in sensor networks, but usually for ECUs inside a car this is not a main concern since controllers do not strongly rely on small batteries. The most critical part, in control systems where this protocol is mostly used, is the authentication delay, i.e., how fast a packet can be deemed as authentic. In particular we must assure that a node, if the bus is not taken by a higher priority message, is able to transmit the message and the message can be checked for authenticity as soon as possible. This condition is initially limited by the computational power (as shown in Chapter 3), but as checking for authenticity can happen only as soon as the disclosure delay expires and the next element of the chain is committed, this also depends on the structure of the chain which is determined by the amount of memory available, and also by the bandwidth. While in sensor networks the disclosure interval is usually in the order of tens or hundreds of milliseconds here we bring this delay lower by 2 to 3 orders of magnitude. Depicting an optimal protocol setting in this context is not straight forward and we study several trade-offs in what follows.

4.2.1 Environment and Protocol Description

As specified in the CAN protocol specification, CAN is a message oriented bus while TESLA appears to be source oriented, i.e., it assures that a message originates from a particular sender. We emphasize that there are many practical scenarios in which the source of a particular CAN message does matter and in practice identifiers are frequently uniquely associated to a particular node. Thus the message oriented nature of CAN should not be interpreted in a strict sense, where the source of the

message is irrelevant. Even for the case of an ID oriented authentication (where authentic messages with the same ID can originate from different nodes) a TESLA like protocol will prove to be more suitable for adding new nodes on the bus since they can authenticate messages via the broadcast scheme without needing to share the secret key for a particular ID.

From an upper view, the design paradigm is the following. Memory, computational speed, bandwidth, initialization time and the synchronization error give bounds on the structure of the chains that we can use. This further bounds the authentication delay, i.e, the delay at which authentication keys arrive on the bus, which is crucial to us as messages cannot be authenticated faster than the disclosure delays. To improve on this delay, we design several variants of the protocol that are presented in section 4.2.2.

All protocol variants use multiple levels of one-way key chains with the structure suggested in Figure 4.1. The relevant notations with respect to the chain structure are: ℓ the number of chain levels, $\sigma_i, i = 1..\ell$ the length of the chain on level i, $\delta_i, i = 1..\ell$ the disclosure delay on level i, ξ the safety margin for releasing authentication packets and δ_{norm} the *normalized disclosure delay* which will be clarified in the next section along with other details. Informally, bullets depict keys from the key chains and the horizontal black arrows denote that they are derived from a previous key. As usual in such protocols, keys are generated and consumed in a reverse order, thus the time arrow on the bus points in opposite direction to the arrows that generate the keys. Packets arriving on the communication bus are depicted as well, the dotted lines from an element of the chain to the packet denotes that the element was used as a key, and for the re-initialization packets in particular one element of the key chain was also used as a message. Packets containing keys are marked by K and commitments, i.e., MAC codes that authenticate forthcoming key chains, are marked by C.

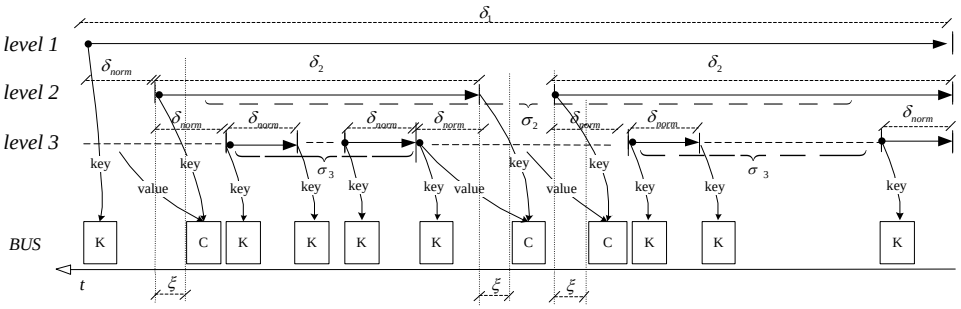


Figure 4.1: Broadcast sequence with normalized time δ_{norm}.

Before the broadcast protocol can run, we need an initialization protocol. Its role is to allow each unit to commit or retrieve its initialization values and to achieve

time synchronization with the sender. This part of the protocol can also rely on more expensive algebraic operations required by public-key cryptography. For example, each principal can authenticate itself by using a public key certificate that is signed by a trusted authority. Initial authentication based on public-key infrastructure can be preferable to assure composability. Thus we require that each node must store the public key of a trusted authority. Although public key certificates are larger and will require more than one frame (which can carry at most 64 bits) in general it should not be a problem to transport them over CAN if this does not happen too often and just in the initialization stage. If public keys are not a feasible alternative, then initialization keys can be hardcoded in an off-line manner.

Time synchronization is done with respect to a central node, which will play the role of a communication master. As usual, synchronization between two nodes is loose and it requires a handshake and counting the round trip time until it is below a tolerance margin. This is usually achieved in two protocol steps as follows: $A \rightarrow B : N_A; B \rightarrow A : Sig_B(t_B, N_A)$. Here N_A denotes a nonce generated by principal A and t_B denotes the current time at principal B when sending its response. Afterwards, the round trip time ϵ_{AB} becomes the synchronization error. If the nonce was sent by A at time t_0 and now A's clock points to t_1 then A knows that the time on B side is in the interval $[t_B + t_1 - t_0,\ t_B + t_1 - t_0 + \epsilon_{AB}]$. However, in our scenario a digital signature costs tens, or hundreds of milliseconds, which will result in an even larger disclosure delay. Because of this, instead of a digital signature we will use a message authentication code which is several orders of magnitude faster. In particular, in our experiments, the round-trip-time was less in the order of several hundreds micro-seconds as shown in the section dedicated to experimental results.

4.2.1.1 Sender's Perspective

For the sender side, we first define the structure of the key chain with respect to each level and then we define the precise timings for the disclosure of each key. We make use of a *timing template* which is used to compute the timings for each level (based on chain lengths and disclosure delays) and a *function template* which is used to generate the keys on each level (based on a one-way function). Different to previous work, we use the *function template* to allow the generation of chains from different levels, with different functions, that will provide good speed-ups in the following variants.

Definition 1. We define the *timing template* as a ℓ-tuple of positive integer pairs denoting the chain length and disclosure delay for each particular level, i.e., $\mathbf{T}_\ell = \{(\sigma_1, \delta_1), (\sigma_2, \delta_2), ..., (\sigma_\ell, \delta_\ell)\}$.

Definition 2. We define the *function template* as a ℓ-tuple of functions that are used to generate the keys on each level, i.e., $\mathbf{F}_\ell = \{F_1, ..., F_\ell\}$.

Definition 3. We define the *indexed key collection* $\mathbb{K}_{\mathbf{T},\mathbf{F}}$ as a tuple of *time-indexed keys* K_τ, i.e., keys entailed by a vector τ with ℓ elements that defines the exact disclosure time for the key. Given *timing template* $\mathbf{T}_\ell$, *function template* $\mathbf{F}_\ell$ a *time-indexed key* is generated as: $K_{\tau_{left}|\tau_i|\overline{0}} = F_i(K_{\tau_{left}|\tau_i+1|\overline{0}}), \forall i \in [1, \ell], \tau_i \in [0, \sigma_i - 1]$.

Here $K_{\tau_{left}|\sigma_i|\overline{0}}$ is the initialization key for the particular key-chain, computed via a key-derivation process from some random fresh material generated at each initialization as $K_{\tau_{left}|\sigma_i|\overline{0}} = KD(K_{\text{rnd}}, \tau_{left})$, K_{rnd} is some fixed random value and KD is a key derivation function. Here τ_{left} is a placeholder for any left part of the index vector τ and the right part, denoted by $\overline{0}$, is always zero.

The previous definition allows the generation of chains on multiple levels with the specified length as suggested in Figure 4.1.

Now we can establish the exact disclosure time for each key. For this we let t_{start} denote the time at which the broadcast was started on the sender side and assuming there are no clock drifts for the sender the exact release time of the keys follows.

Definition 4. Let $\mathcal{DT}(K_\tau, \mathbf{T}_\ell)$ denote the disclosure time of the *indexed key* K_τ based on *timing template* $\mathbf{T}_\ell$. Given a broadcast started at t_{start} the disclosure time of K_τ is: $\mathcal{DT}(K_\tau, \mathbf{T}_\ell) = t_{start} + \sum_{i=1..\ell}(\delta_i \cdot \tau_i)$.

4.2.1.2 Receiver's Perspective

We consider the case of a receiver $\mathcal{R}$ and sender $\mathcal{S}$ with synchronization error $\epsilon_{\mathcal{S},\mathcal{R}}$. Now we define the security condition that must be met by all packets that contain authentication information, i.e., MAC codes, produced with an indexed key K_τ.

Definition 5. Given synchronization error $\epsilon_{\mathcal{S},\mathcal{R}}$ and $t_{\mathcal{S}}$ the time value reported by $\mathcal{S}$ on a synchronization performed at t_{sync} with $\mathcal{R}$, the minimum and maximum time on the $\mathcal{S}$'s side, *estimated* by $\mathcal{R}$ having local clock pointing at $t_{\mathcal{R}}$ are: $\mathcal{ET}_{\min}(t_{\mathcal{R}}) = t_{\mathcal{R}} - t_{sync} + t_{\mathcal{S}}$, $\mathcal{ET}_{\max}(t_{\mathcal{R}}) = t_{\mathcal{R}} - t_{sync} + t_{\mathcal{S}} + \epsilon_{\mathcal{S},\mathcal{R}}$.

Definition 6 (Security Condition). Given *timing template* $\mathbf{T}_\ell$, an authentication packet computed with K_τ received at time $t_{\mathcal{R}}$ must be discarded unless: $\mathcal{ET}_{\max}(t_{\mathcal{R}}) \leq \mathcal{DT}(K_\tau, \mathbf{T}_\ell)$.

This condition ensures that an authentication packet is not accepted after the authentication key was already disclosed.

4.2.1.3 Generic Description of the Protocol

The generic description of the protocol now follows. We underline that this description does not include particular optimizations that are presented in the section dedicated to practical variants. It works only as a high level description for the forthcoming protocols.

Definition 7. Given *indexed key collection* $\mathbb{K}_{\mathbf{T},\mathbf{F}}$ and two roles called sender and receiver denoted by $\mathcal{S}$ and $\mathcal{R}$ each having the synchronization error $\epsilon_{\mathcal{S},\mathcal{R}}$, protocol $Broadcast_{\mathcal{S},\mathcal{R}}[\mathbb{K}_{\mathbf{T},\mathbf{F}}]$ is defined by the following two rules for the two roles: i) $\mathcal{S}$ sends K_τ at $\mathcal{DT}(K_\tau)$ and $MAC(K_\tau, M)$ in any empty time-slot with the condition that $MAC(K_\tau, M)$ is released no latter than $\mathcal{DT}(K_\tau) + \xi$. Message M can be released at any time, ii) $\mathcal{R}$ discards all $MAC(K_\tau, M)$ received at $t_{\mathcal{R}}$ for which $\mathcal{ET}_{\max}(t_{\mathcal{R}}) \leq \mathcal{DT}(K_\tau, \mathbf{T}_\ell)$ does not hold and deems authentic all other messages for which $MAC(K_\tau, M)$ can be verified with an authentic key. A key $K_{\tau_{left}|\tau_i|\overline{0}}$ is authentic if and only if $K_{\tau_{left}|\tau_i|\overline{0}} =$

$\mathcal{F}(K_{\tau_{left}|\tau_i+1|\bar{0}})$ and $K_{\tau_{left}|\tau_i+1|\bar{0}}$ is a previously received authentic key (note that $K_{\tau_{left}|0|\bar{0}}$ must be committed via a MAC).

Here ξ denotes a tolerance margin for the time at which a MAC with a particular key can be sent. Indeed, sending MACs too close to the disclosure time may be useless because the receiver may have to discard them if the security condition cannot be met. Thus ξ must be fixed as initial setup parameter for the protocol. In time interval $[\mathcal{DT}(K_\tau), \mathcal{DT}(K_\tau) + \xi]$ the sender can safely disclose any kind of data packet, but not MACs.

$Broadcast_{\mathcal{S},\mathcal{R}}[\mathbb{K}_{\mathbf{T},\mathbf{F}}]$ is a secure broadcast authentication protocol. The security of this family of protocols is well established, formal proofs of security can be found in [13] and [101]. The informal argument is that from $MAC_{\mathcal{K}}(M)$ and $F(\mathcal{K})$ an adversary cannot produce $MAC_{\mathcal{K}}(M')$ for any $M' \neq M$ since $\mathcal{K}$ is not known as well as it cannot be found from $F(\mathcal{K})$. By the time $\mathcal{K}$ is released it is already too late for the adversary to send a *MAC* and a message as they will not be anymore accepted by the receiver due to the time constraint. A more formal proof sketch can be done by using random oracles. It is commonly acknowledged that although random oracles do not give an absolute proof they can be used at least as a sanity check to prove the security of protocols. If we assume that oracle $\mathcal{O}^F$ that computes function F can be replaced by a random oracle $\mathcal{O}^R$, which outputs k bits, the proof is straight forward. Assume that the adversary has witness polynomially many queries $p(k)$ to oracle $\mathcal{O}^R$. Suppose at some point the adversary is forced to produce $MAC_{\mathcal{K}}(M_{Adv})$ for some message of its choice. The adversary knows just $\mathcal{O}^R(\mathcal{K})$ which is the output of the random oracle and $\mathcal{K}$ is unpredictable subject to the fact that it may have been already asked by the adversary to $\mathcal{O}^R$. This means he can guess it and produce a valid *MAC* only with probability $1/(2^k - p(k))$ - which is negligible.

4.2.1.4 Efficiency parameters

The efficiency of the protocol can be evaluated with respect to memory, CPU and bandwidth. This evaluation has to be done over the entire time horizon of the protocol which can be divided in two parts: initialization time T_{init} and runtime T_{run}. However, bus loads and CPU utilizations, that are going to be defined next, are more relevant only over T_{run} as it is natural to expect that during T_{init} the initialization can takeover the entire bus and CPU but only for a very short period of time. We will use the following notations: *MEM*, *CPU*, *BUS* and their capacities are depicted in the number of keys that can be stored, computed or sent.

For all of these notations, a subscript indicates whether they refer to the initialization stage or the runtime stage. Thus CPU_{init} refers to the amount of work during initialization and CPU_{run} during runtime. By MEM^{total}, CPU^{total} and BUS^{total} we refer to the total available computational power and bus capacity during the entire run-time of the protocol - we use these measures to define CPU and bus loads during runtime. To indicate whether a resource is needed for computing keys or commitments through MAC codes we use *key* and *com* as superscripts.

Definition 8. Given *key collection* $\mathbb{K}_{\mathbf{T},\mathbf{F}}$ we let $\| \mathbb{K}_{\mathbf{T},\mathbf{F}}[i] \|$ denote the total number of keys on level i disclosed during protocol lifetime and $\langle\!\langle \mathbb{K}_{\mathbf{T},\mathbf{F}}[i] \rangle\!\rangle$ the number of key chains from level i.

Obviously we have $\| \mathbb{K}_{\mathbf{T},\mathbf{F}}[i] \| = \sigma_i \cdot \langle\!\langle \mathbb{K}_{\mathbf{T},\mathbf{F}}[i] \rangle\!\rangle$ since the total number of keys is the number of chains multiplied with the chain length. We will use both notations, although it is easy to derive one from the other, in order to make the following relations more intuitive.

Definition 9. Let $\{(c_0, m_0), (c_1, m_1), ..., (c_\ell, m_\ell)\}$ define the CPU and memory requirements for all elements of the *function template* $\mathbf{F}$. For protocol $Broadcast_{\mathcal{S},\mathcal{R}}[\mathbb{K}_{\mathbf{T},\mathbf{F}}]$ we define the following overheads caused by key-chains:

$$MEM^{key}_{init} = MEM^{key}_{run} = \sum_{i=1,\ell} \sigma_i \cdot m_i \tag{4.1}$$

$$CPU^{key}_{init} = \sum_{i=1,\ell} \sigma_i \cdot c_i \quad CPU^{key}_{run} = \sum_{i=2,\ell} c_i \cdot (\| \mathbb{K}_{\mathbf{T},\mathbf{F}}[i] \| - \sigma_i) \tag{4.2}$$

$$CPU^{com}_{init} = \sum_{i=1,\ell} c_i \quad CPU^{com}_{run} = \sum_{i=2,\ell} c_i \cdot (\langle\!\langle \mathbb{K}_{\mathbf{T},\mathbf{F}}[i] \rangle\!\rangle - 1) \tag{4.3}$$

$$BUS^{key}_{run} = \sum_{i=1,\ell} m_i \cdot \| \mathbb{K}_{\mathbf{T},\mathbf{F}}[i] \| \tag{4.4}$$

$$BUS^{com}_{init} = \sum_{i=1,\ell} m_i \quad BUS^{com}_{run} = \sum_{i=2,\ell} m_i \cdot (\langle\!\langle \mathbb{K}_{\mathbf{T},\mathbf{F}}[i] \rangle\!\rangle - 1) \tag{4.5}$$

Equation 4.1 gives the memory requirements which is the sum of the lengths of the chains. In the case of memory there are no variations during initialization and runtime. More, we do not need additional memory to store commitments on the sender as commitments can be sent as soon as they are computed.

Equation 4.2 gives computational time required for keys at runtime and initialization. In the initialization one chain is computed on each level. At runtime, there are $\langle\!\langle \mathbb{K}_{\mathbf{T},\mathbf{F}}[i] \rangle\!\rangle$ key-chains on each level, and each of them has to be computed except the first one which was computed during initialization which gives $CPU^{key}_{run} = \sum_{i=2,\ell} c_i \cdot \sigma_i \cdot (\langle\!\langle \mathbb{K}_{\mathbf{T},\mathbf{F}}[i] \rangle\!\rangle - 1)$. Replacing $\sigma_i \cdot \langle\!\langle \mathbb{K}_{\mathbf{T},\mathbf{F}}[i] \rangle\!\rangle$ with $\| \mathbb{K}_{\mathbf{T},\mathbf{F}}[i] \|$ we get the claimed number of keys computed at runtime.

In Equation 4.3 commitments are measured: one commitment on each level during initialization, and later for each chain on each level (except for the first one which was committed during initialization) one commitment is needed.

Bus requirements for keys during runtime is given in Equation 4.4. All keys from all levels are sent on the bus, while there are no keys (just commitments) sent during

initialization. Commitments are given in Equation 4.5. One chain on each level is committed in the initialization, and later at runtime all chains are committed except for the first one, same as in the case of computational requirements.

To complete the view on efficiency, we should also define the CPU and bus loads over the entire lifetime of the protocol.

Definition 10. Given $RES \in \{MEM, CPU, BUS\}$, $state \in \{run, init\}$ we define the protocol overheads as:

$$RES^{load}_{state} = \frac{RES^{key}_{state} + RES^{com}_{state}}{RES^{total}_{state}} \tag{4.6}$$

One can add to these the overhead induced by the message authentication codes associated to each data packet that is sent over the bus. This is however application dependent, not protocol dependent as in some applications the size of the data packets can be small, and thus adding a MAC to each data packet will greatly increase the overhead while in other applications it may be the reverse, and data packets can be large and the MAC will not significantly increase the overhead. Thus, this measure can be done only on practical case studies.

4.2.2 Practical Variants

Now we discuss practical variants of the main scheme. We proceed with the analysis of the single and multi master case. Then go to the equidistant timing scheme, which provides uniform delays on the bus, and improve it by using reduced variants of hash functions in two schemes that provide ad hoc security in order to minimize the overhead and delays.

Obtaining a variant that is adequate, possibly optimal, for practical use means to satisfy the constraints of the environment. The *generic calibration criteria* for the scheme parameters is the following. Having fixed T_{run} and δ_{norm} we determine chain structure (lengths and levels) and timings which give $\mathbf{T}_\ell$, $\mathbf{F}_\ell$ and $\mathbb{K}_{\mathbf{T},\mathbf{F}}$. Then, we determine bus, CPU and memory loads for comparison.

4.2.2.1 The Multi Master and Single Master Case

CAN must allow each node to be a potential sender. The case of k senders can be easily derived from the previous formalism. We can multiplex the senders by using δ_{next} which we call the *next sender delay*. By this, we can modify the disclosure timings to $\mathcal{DT}_k(K_\tau) = \mathcal{DT}(K_\tau) + k \cdot \delta_{next}$ and the security condition accordingly for the case of k senders.

However, allowing more than one sender will result in a bus that is heavily loaded by keys and commitments. To avoid this, having only one communication master is preferable. In the case when one of the slave nodes needs to broadcast authentic information it can perform a request to the communication master under the assumption that each slave node shares a secret key with the master that can be used for

authentication. We can take advantage of the CAN nature as each slave can place its data frame on the bus, along with a message authentication code computed with the shared key. The communication master can verify this MAC and, if correct, it will send a next frame that contains the broadcast authentication information, i.e., the MAC with the current key according to the broadcast protocol. The rest of the protocol is unchanged, we do not give a methodology to compute parameters as this is going to be discussed for the next variants. This approach will not increase the authentication delay if the slave nodes are able to send the message and its MAC during the same disclosure period in which the master can continue to broadcast the authentication MAC.

4.2.2.2 Equidistant Timing (Delayed) Authenticated CAN (ETA-CAN)

For practical reasons, a solution which assures a uniform bus load is preferable. This is mostly because packets carrying data must be delayed until all keys and commitments are sent since they have priority on the bus (otherwise the protocol will succumb and have to be re-initialized).

For this, we use a procedure which we call *equidistant timing* by which all keys are disclosed at moments of time separated by equal delays. This is relevant also because we can use upper layer chains not only to authenticate the commitments of keys from the lower levels but also to authenticate information packets as well. The same *equidistant* release will be used for key commitments. Thus, we will normalize the disclosure time on the last level and then compute the disclosure delays on the upper levels. These disclosure timings are suggested in Figure 4.1.

Definition 11. For the ETA-CAN we define the disclosure delays as:

$$\delta_\ell = \delta_{norm} = \frac{T_{run}}{\sum_{i=1,\ell} \| \mathbb{K}_{\mathbf{T},\mathbf{F}}[i] \|} = \frac{T_{run}}{\prod_{i=1,\ell}(\sigma_i + 1) - 1} \tag{4.7}$$

$$\delta_i = \delta_{norm} \cdot \prod_{j=i+1,\ell} (\sigma_j + 1), 1 \leq i < \ell \tag{4.8}$$

It is easy to note that given a fixed amount of memory which must accommodate the chains and a fixed number of levels ℓ, the disclosure delay δ_{norm} and the overheads for CPU and bus have an inverse variation. Thus: the minimal disclosure delay is achieved if chains are of equal size while the minimal computational and communication overhead is achieved if upper level chains are smaller. This is intuitively since the product of two values whose sum is fixed is maximal if the two values are equal and minimal if one of the values is 1.Note that if $x + y = z$ then $\forall k \geq 1, z/2 \cdot z/2 > (z/2 - k) \cdot (z/2 + k) = z^2/4 - k^2$. To achieve the minimum delay, the product of the chain lengths $\prod_{i=1,\ell}(\sigma_i + 1)$ must be maximal while the sum of these values is restricted to the amount of available memory. If we split this product exactly into the half left and half right terms, assuming an even number of terms which is without loss of generality, then the maximum product is achieved if: the left and right

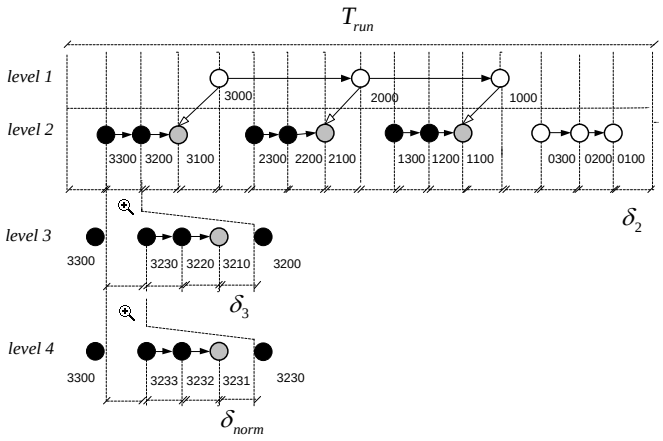


Figure 4.2: Chain structure for the *BETA* scheme with $\ell = 4, \sigma_{norm} = 3, \delta_{norm} = T_{run}/255$.

are maximal and they are equal, and so on. On the contrary, having upper chains of smaller size means less commitments since each element from the upper chains commits one chain on all lower levels. Also, in general, the authentication overhead will increase with the number of levels for the same reason.

4.2.2.3 Balanced Equidistant Timing delayed Authenticated CAN (BETA-CAN)

Based on the previous remark on efficiency we explore the variant with chains of equal sizes on all levels. To clarify previous notations Figure 4.2 shows an example of chain structure for the case of $\sigma = 3$ (note that the same key-chain size is on all levels).

Since the entire run-time of the protocol is $T_{run} = \delta_{norm} \cdot [(\sigma+1)^{\ell^{BETA}} - 1]$ the number of levels follows as:

$$\ell^{BETA} = \left\lceil \log_{\sigma+1}\left(\frac{T_{run}}{\delta_{norm}} + 1\right)\right\rceil \tag{4.9}$$

The disclosure delay of the last level is δ_{norm} while for the upper levels the delay can be computed as:

$$\delta_i^{BETA} = \delta_{norm} \cdot (\sigma + 1)^{\ell^{BETA} - i} \tag{4.10}$$

Having these defined the performance indicators for memory, CPU and bus can be derived. These indicators are summarized for all variants in Table 4.1.

Figure 4.3 shows the influence of chain length and levels on various parameters.

Plots are taken for a time range T_{run} = 24 hours while the bus speed is approximated to about 6000 packets per second. In the case of variations with chain lengths, plots (i) and (ii) are given for three and four levels of key chains. We note that the delays drop rapidly by increasing the number of levels in plot (i), but in the same manner the overhead increases (ii) (at 100% the bus is locked and communication halted). Plot (iii) shows the variation of memory requirements, which is the same as the initialization time, and plot (iv) of commitments with the number of chain levels. The same drop of memory requirements and increase in commitment packets can be seen. For plots (iii) and (iv) the delay is fixed to 5 ms.

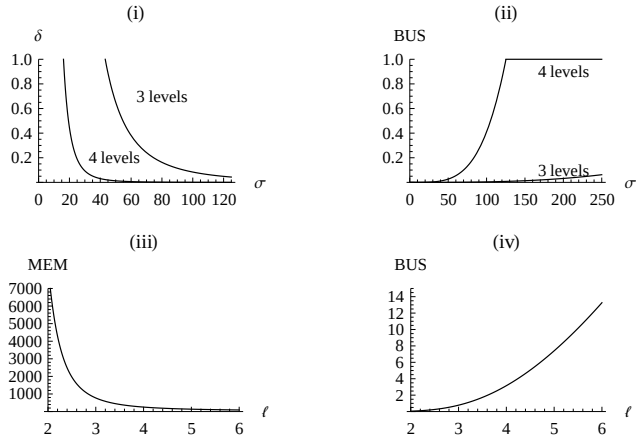


Figure 4.3: Various overheads and delay variation with length or levels: (i) disclosure delay , (ii) overhead caused by keys and commitments on the bus, (iii) keys stored in memory and (iv) commitments on the bus (per second).

	BETA-CAN	Ad-BETA-CAN	Ad-GETA-CAN
$MEM^{\star}_{key}$	$\ell \cdot \sigma \cdot m$	$\sigma \cdot \sum_{i=1,\ell} m_i$	$\sum_{i=1,\ell-1} m_i + \sigma_\ell \cdot m_\ell$
CPU^{init}_{key}	$\ell \cdot \sigma \cdot c$	$\sigma \cdot \sum_{i=1,\ell} c_i$	$\sum_{i=1,\ell-1} c_i + \sigma_\ell \cdot c_\ell$
CPU^{init}_{com}	$\ell \cdot c$	$c_0 + \sum_{i=1,\ell-1} c_i$	$c_0 + \sum_{i=1,\ell-1} c_i$
BUS^{init}_{com}	$\ell \cdot m$	$m_0 + \sum_{i=1,\ell-1} m_i$	$m_0 + \sum_{i=1,\ell-1} m_i$
CPU^{run}_{key}	$[(\sigma+1)^{\ell} - \sigma \cdot \ell - 1] \cdot c$	$\sigma \cdot \sum_{i=2,\ell} [(\sigma+1)^{i-1} - 1] \cdot c_i$	$\sum_{i=2,\ell-1} (2^{i-1} - 1) \cdot c_i + (2^{\ell-1} - 1) \cdot \sigma_\ell \cdot c_\ell$
BUS^{run}_{key}	$[(\sigma+1)^{\ell} - 1] \cdot m$	$\sigma \cdot \sum_{i=1,\ell} (\sigma+1)^{i-1} \cdot m_i$	$\sum_{i=1,\ell-1} 2^{i-1} \cdot m_i + 2^{\ell-1} \cdot \sigma_\ell \cdot m_\ell$
CPU^{run}_{com}	$[\frac{(\sigma+1)^{\ell}-1}{\sigma} - \ell] \cdot c$	$\sum_{i=2,\ell} [(\sigma+1)^{i-1} - 1] \cdot c_i$	$\sum_{i=2,\ell} (2^{i-1} - 1) \cdot c_i$
BUS^{run}_{com}	$[\frac{(\sigma+1)^{\ell}-1}{\sigma} - \ell] \cdot m$	$\sum_{i=2,\ell} [(\sigma+1)^{i-1} - 1] \cdot m_i$	$\sum_{i=2,\ell} (2^{i-1} - 1) \cdot m_i$

Table 4.1: Overheads at initialization and run-time for *MEM*, *CPU* and *BUS*

4.2.2.4 Ad hoc secure Balanced levels ETA-CAN (Ad-BETA-CAN)

To increase performance we will use reduced versions of hash functions. The following definition is informal and will serve only as a heuristic for the security of the schemes.

We call function F_k *ad hoc secure* with respect to time interval δ if given $y = F_k(x)$, with $F_k(x)$ publicly known, it is infeasible to find an x' in time δ such that $y = F_k(x')$ (note that x' does not necessary need to be same as x since any x' which has the same image under F_k will suffice to break the protocol). Here k plays the role of an index for function F. More concrete, in our practical implementation we use for function F_k truncated versions of hash functions in order to reduce the overhead on the bus. However, since the image of F is reduced, for example to 32 bits in the worst case, it can be feasible for an adversary to mount a pre-computed dictionary attack. To avoid this, we compute the α-bit truncated hash chain in the following manner: at each step we compute $K_{\tau_{left}|\tau_i|\bar{0}} = \lfloor F_k(kd||K_{\tau_{left}|\tau_i+1|\bar{0}})\rfloor_\alpha$. Here kd stands from some material derived from the key template, i.e., previously released keys, in order to assure sufficient entropy against pre-computed attacks, similar to salting. Note that the same truncation can be done for MAC codes under the restriction that messages and keys are not released later than the security lifetime δ (if the message or key is to be released later, then the appropriate size for the MAC is to be chosen).

Note that the disclosure delays are now needed to establish the exact security level that must be met by functions on each level of the chains. Therefore, these delays determine the *function template* that can be used. For reduced variants of the hash functions, only heuristic arguments can be given, that is, protocol $Broadcast_{\mathcal{S},\mathcal{R}}[\mathbb{K}_{\mathbf{T},\mathbf{F}}]$ is *ad hoc secure* with respect to the corresponding disclosure timings.

Same as in the previous scheme, we use chains of equal sizes to minimize the number of layers but we select different functions on each level such that the function is *ad hoc* secure with respect to the disclosure delay on the particular level. Thus, given δ_{norm} we first select the less intensive function that is *ad hoc secure* with respect to δ_{norm}. Then we assume that this function is going to be used on all levels, take the constraints on memory and CPU and successively try the smallest value of ℓ until they are satisfied. Then, we proceed from the ℓ-th chain upward to change the function to one that is *ad hoc secure* with the respective delay. If the constraints are not fulfilled we chose a bigger ℓ and so on. The number of levels and the delays are computed in the same manner as previously while performance indicators are summarized as well in Table 4.1. An explanatory example follows after the introduction of the next scheme.

4.2.2.5 Ad hoc secure Greedy last level ETA-CAN (Ad-GETA-CAN)

This variant uses a greedy strategy in order to minimize memory overheads. Given δ_{norm} and T_{init} we first select the less intensive function that is ad hoc secure with respect to δ_{norm}. Then we use the entire T_{init} time to compute a chain from the last level, subject only to memory constraints, i.e., if memory exhaust before T_{init} we stop. For example, given σ_ℓ the length of the chain on the last layer we choose the number

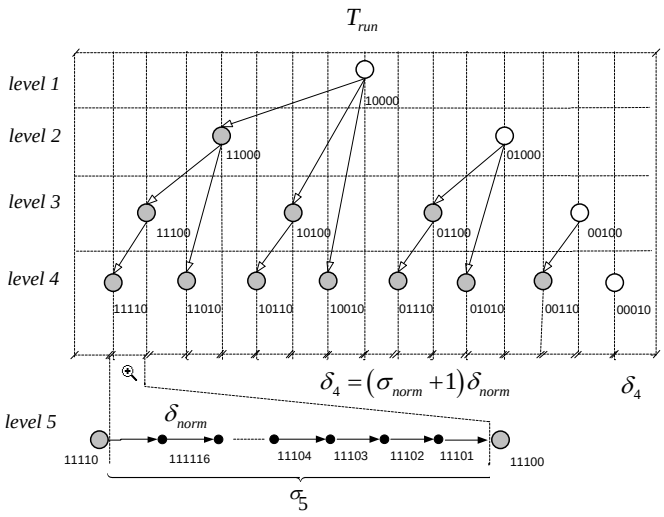


Figure 4.4: Chain structure for the *GETA* scheme with $\ell = 5, \sigma_{norm} = 16, \delta_{norm} = T_{run}/255$.

of levels ℓ. Then we set to 1 the length of each chain from level 1 to $\ell - 1$. In this way we minimize the memory and computational time for the upper layers. Since the number of upper layers is maximum, due to the reduced chain length, this also increases initialization overhead on the bus. T_{init} should be only slightly overloaded since the initialization time is minimum when the number of levels is maximum. If memory is also exhausted by the first layer, we cut down as many elements as are needed to fit the upper layers.

Figure 4.4 depicts the structure of the chains in this case while an explanatory example follows in the next subsection. The parameters of the scheme can be computed as follows. Having $T_{run} = \delta_{norm} \cdot [2^{\ell^{GETA}-1} \cdot (\sigma + 1) - 1]$ the number of levels and the disclosure delay are:

$$\ell^{GETA} = \left\lceil \log_2 \frac{T_{run} + \delta_{norm}}{\delta_{norm} \cdot (\sigma + 1)} \right\rceil + 1 \tag{4.11}$$

$$\delta_i^{GETA} = \delta_{norm} \cdot 2^{\ell^{GETA}-i-1} \cdot (\sigma + 1) \tag{4.12}$$

Note that the same relations hold for the GETA scheme with or without ad hoc security. For simplicity, in the discussion that follows we will omit the prefix *ad* whenever it is clear from the context whether we refer to the standard scheme or to the ad hoc secure scheme.

4.2.2.6 Comparison and limitations

We now give an explanatory example for the *ad hoc secure* schemes. Consider the following truncated versions of the MD5 hash function with the corresponding security level $\mathcal{SL}$ and performance descriptor $\mathcal{PD}$:

$$\mathcal{SL}: \begin{pmatrix} \lfloor MD5 \rfloor_{32} & 10^{-3} \\ \lfloor MD5 \rfloor_{48} & 1 \\ \lfloor MD5 \rfloor_{64} & 10^{3} \\ \lfloor MD5 \rfloor_{128} & \infty \end{pmatrix} \mathcal{PD}: \begin{pmatrix} 32 & 11 \times 10^{-6} \\ 48 & 11 \times 10^{-6} \\ 64 & 11 \times 10^{-6} \\ 128 & 11 \times 10^{-6} \end{pmatrix}$$

That is, for ad hoc security with respect to a delay of 10^{-3} seconds a truncated version of MD5 can be used, requiring only 32 bits, etc. The computational timings are according to the experimental results from the next section and they serve here as a calibration example. Note that computational speed is the same for all truncated versions and the gain is only in the bus load and memory requirements.

Let us fix for our example the disclosure delay to $\delta_{norm} = 1ms$ with a one year runtime of the protocol $T_{run} = 31 \times 10^6 s$ and $T_{init} = 1s$.

With the *GETA* scheme, by using computational power as restriction we can compute up to 90.000 elements in T_{init} which is far too much for the memory, so we limit σ_ℓ to 1000 which is reasonable to fit in memory and would ease the computation. This gives $\ell = 26$ with length 1000 on level 26 and 1 on the other levels. The disclosure delay on level 26 is $10^{-3}s$ so we can use the 32 bit version. On level 25 the disclosure delay is around $2s$ so we skip to the 48 bit version which can be used up to level 23 that has a delay of $8s$. Levels up to 16 can use the 64 bit version and the rest of the levels will use the 128 bit version. The overall memory load is $1000 \times 32 + 3 \times 48 + 7 \times 64 + 15 \times 128 = 34512$ bits. The commitments on the bus reach up to 11×10^9 bits in a year.

With the *BETA* scheme, by fixing $\sigma = 1000$ we get $\ell = 4$ but this is due to the ceiling which increased the value from the actual 3.5. With chains of 1000 elements this will increase T_{run} too much and will require too much memory as well. Note that the delay on level 4 is 10^{-3} and it increases on each level with a factor of 1000, which means that roughly level 1 uses the 128 bit version, level 2 the 64 bit version, level 3 the 48 bit version and level 4 the 32 bit version. For chains of length 1000 this gives a memory load of $1000 \times 32 + 1000 \times 48 + 1000 \times 64 + 1000 \times 128$, that is 272000 bits which is almost 8 times more than for the *GETA* scheme but T_{run} has also increased about 30 times. By empirical test we get that $\sigma = 428$ will not change T_{run} and will require only 116416 which is just 3 times more than for *GETA* scheme. For the commitments on the bus its the reverse as they reach up to 3.7×10^9 bits in a year which is 3 times less than for *GETA*. Having a lighter bus load should be preferable for practical applications, therefore *BETA* seems to be better.

We note that larger chains can be stored if time-memory trade-offs are used. Such trade-offs, based on storing only some elements of the chain from which the rest of the chain is rebuilt, were extensively studied in [65], [25], [35], [58]. Still,

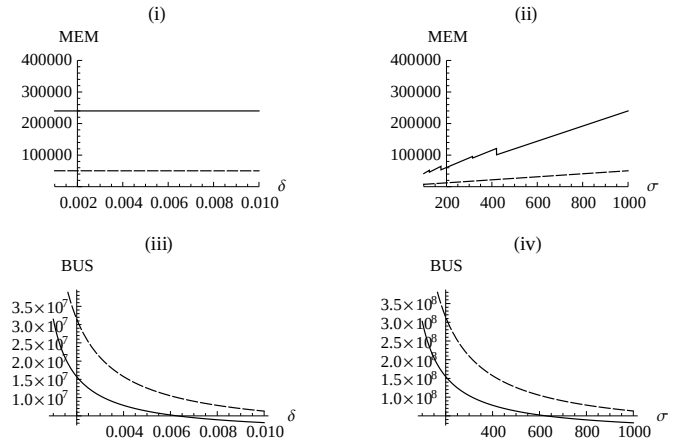


Figure 4.5: Comparison between *BETA* (continuous line) and *GETA* (dotted line) schemes at MEM and BUS requirements for: fixed σ = 1000 and variable δ_{norm} in (i) and (iii), variable δ_{norm} and fixed σ = 1000 in (ii) and (iv).

so far we refrained from using them since this will load the CPU which leads to an implementation that is not energy efficient. However, studying such a trade-off can be subject of future work.

Figure 4.5 shows the generic difference between the *BETA* and *GETA* schemes assuming fixed delay and variable length or the reverse. As already shown in the previous example, the relevant aspect is that *BETA* gives a lighter bus due to fewer chain commitments. However, *GETA* is much better in terms of memory requirements, also a relevant constraint of our environment.

4.2.2.7 Security considerations for the ad hoc secure scheme

It is not easy to define rigorous parameters for ad hoc security. Here we used 32 bits as the minimum recommended for authentication tags against "real-time" attacks by the most recent ECRYPT recommendations on key sizes [59]. This security level is certainly enough against common equipments, such as standard CPU's, that can perform hashes in the order of 10^6 per second. Dedicated equipments, built on FPGAs and more recently on GPU accelerators, can reach billions, i.e., 10^9, of crypto-operations per second which is close to the 32 bit security bound. However, such equipments are not cheap, their cost is usually in the order of tens of thousands of dollars and thus they are not available to average adversaries. Even for the case of such adversaries, with dedicated hardware and high financial resources, we believe that 48 bits should still be infeasible to invert in less than a second.

These considerations are helpful to asses the security of the protocol from a quantitative perspective. Further, for validating the security of large scale systems, where such protocols can be part of, automatic verification with specialized tools is an alternative. Using such techniques has become more frequent in industrial systems, a

case study on fieldbus is available in [24].

4.2.2.8 Synchronization issues

The synchronization error achieved by the loose time synchronization mechanism is small enough to accommodate the disclosure delays of the protocol. Even for the smallest disclosure delay in our practical implementation this error is still less than half of it. Indeed, this can be further improved by using more specialized protocols. A good start point could be the Precision Time Protocol (PTP) but this protocol is not cryptographically secure so further developments would be necessary.

Clock drifts between oscillators can lead to more frequent resynchronizations. Common recommendations for CAN bit timings are a minimum oscillator tolerance of 1.49% at 125 kbps and 0.49% at 1 Mbps. These tolerances are enough to eliminate frequent resynchronization for most disclosure delays. However, if we push the disclosure delays to their lower limit this can become a relevant issue.

We consider an example to clarify this. Around $300\mu s$ is the minimum delay achieved in our application setting for a 1Mbps CAN bus. A tolerance of 0.49% will result in a maximum of $2 \times 0.49\% \approx 1\%$ drift between the sender and the receiver (considering the worst scenario in which oscillators drift apart in opposite directions). This means that at each $300\mu s$ the receiver clock drifts with $3\mu s$ which means that after 100 packets the receiver will either drop all subsequent packets (if its clock is faster), or an adversary may be able to forge packets (if its clock is slower). However, such a clock drift results from using oscillator tolerances that are quite at the edge. Let us emphasize that in our practical setting the two Infineon TriCore controllers had a drift of around 2.73 seconds after 24 hours when running at maximum speed. This leads to a drift of $3.15ns$ at each $100\mu s$ and even after 1 second of broadcast the drift is around $3\mu s$ which will keep the protocol secure if we set $\xi = 3\mu s$. Thus, the first and the most natural solution is to use better oscillators which are available on the market and already present in most of the devices. In case they are not available in a particular setting, higher authentication delays should be considered.

Nevertheless, resynchronization can and should be performed in a more efficient manner by using the broadcast protocol as long as the node is not yet completely desynchronized. Thus, nodes that have weak oscillator tolerances must send a resynchronization request by placing a nonce on the bus and the sender will answer with the usual $MAC_K(time, nonce)$ which will be authenticated with the forthcoming key K released in the broadcast protocol. More concrete, for security reasons, we enforce such a resynchronization to be performed when it is expected that the clock of the sender and receiver drifted by ξ. This leads in our practical scenario to send a resynchronization packet in the worst case at each minute (depending on the exact disclosure delay, safety margin and microcontroller speed). Also note that for efficiency synchronization can be in this way processed for more than one receiver at once in the sense that different nonces from different receivers can be merged by the broadcast master in one response. Care should be taken since this will increase the

synchronization error while it is still mandatory for each receiver to send its own fresh nonce. The resynchronization procedure shouldn't raise scalability problems if it is not triggered too often.

4.2.2.9 Coexistence with other traffic

The influence of the broadcast traffic on already existent real-time CAN bus traffic and vice-versa is a relevant topic. First, let us underline that from the authentication point of view, timing is critical only for keys and MAC codes (keys are disclosed at precise moments and MACs must be released no later than the corresponding key), while the authenticated messages can be released at any point. There are two cases that need to be considered.

In the first case, the broadcast authentication traffic has higher priority than the real time real-time traffic existent on the bus (which can be disturbed in this way). By construction of the equidistant timing protocol, we assure that the keys are uniformly distributed over the life-time of the protocol. Thus, a scenario with periods of burst, with more keys than usual on the bus, will not occur. The only burst may be due to MACs that also need to be sent before the keys are disclosed. To avoid disturbance of the real time traffic on CAN, we enforce the use of the same priority for the MAC as for the message they authenticate, and not a higher one. Thus, real-time constraints will not be violated - each MAC has the same priority as the message it authenticates.

In the second case, the broadcast authentication protocol has lower priority than some of the existent traffic. Indeed, it is not our main intention to use the protocol in such scenarios, but tweaks in the protocol are possible to accommodate it with already existent higher priority traffic. For this case we propose the following strategy concerning the keys and the MACs. As MACs cannot be released later than the keys but their precise release time is not critical, they will be released as quick as possible (roughly short after the release of the previous key) which gives higher chances for a MAC to be released in time. For the keys however, as premature release is not feasible since this will change the timings of the scheme, they will be released at the precise time they are scheduled or as soon as the bus is free afterwards. Note that releasing the keys later doesn't cause any insecurity. An exception is the ad hoc variant were the security level of the key and MACs should be chosen according to the expected delays. For safety reasons, in the case of lower priority for the broadcast protocol, one can also enforce a key-recovery procedure. As re-initialization of the protocol will be too expensive, for this purpose one can use safety-chains, i.e., another upper layer of key-chains that are computed a-priori by the sender to allow recovery after an error has occurred. This allows a trade-off between traffic priority and computational power.

Indeed, in the case of a heavily loaded bus, disturbances between different types of traffic are unavoidable. In such situations, physical separation between the two kinds of traffic may be the best alternative.

4.2.3 Experimental Results

A proof-of-concept implementation was done in order to determine the behavior of the proposed solution in a real environment. Our test setup consists of a master node and multiple slave nodes. The communication master holds the key chains which are used to send authenticated messages to slave nodes.

We made tests on Freescale S12X and Infineon TriCore, two different classes of microcontrollers used in automotive and industrial applications. Detailed controller specifications were presented in Chapter 3.4.1. When using the TriCore platform, the TC1797 was used as the maser while TC1782 served as the slave node. For timing issues, we used the 56 bit free-running timer available on TriCore. Running at the highest speed, with one tick every 11.11 ns, this timer will overflow after 25.39 years which is enough to fit the life-time of many industrial devices and automotive products.

4.2.3.1 Computational performance

To evaluate protocol performance we have to start from the performance of the microcontrollers for computing cryptographic primitives. In addition to the results presented in Chapter 3.4.2, Table 4.2 holds actual execution times on our microcontrollers. The SHA-512 reference implementation is based on 64 bit variables which are not supported by the S12 compiler. As adapting this implementation to fit the S12 compiler would lead to an additional decrease in performance we opted not to evaluate the execution speed for this function. The S12X results were obtained for a frequency of 40 MHz (when overclocking at 80 MHz the execution speed is doubled) while TriCore microcontrollers were running at 180 MHz. The two TriCore microcontrollers have the same core version thus identical performances. These measurements show that on average the primitives were performed approximately 2.12 times faster on XGATE than on S12X while the TriCore implementations are, as expected, much faster (1 to almost 2 orders of magnitude).

Length (bytes)	S12X execution time (μs)			XGATE execution time (μs)			TriCore execution time (μs)			
	MD5	SHA-1	SHA-256	MD5	SHA-1	SHA-256	MD5	SHA-1	SHA-256	SHA-512
0	732	2285	5510	312.5	1000	3155	10.16	17.27	36.75	119.2
1	736	2290	5520	314.5	1002	3155	10.58	17.23	36.42	122.2
8	737	2290	5520	314.5	1002	3155	11.77	17.22	36.18	122.0
16	738	2290	5500	316.0	1004	3150	11.77	17.23	35.35	121.0
32	739	2295	5490	317.5	1004	3145	11.77	17.23	34.58	120.4
64	1414	4510	10860	605	1976	6240	16.56	25.82	64.39	117.6

Table 4.2: Performance of S12X, XGATE and TriCore in computing cryptographic primitives.

The computation speed, memory and the low speed CAN transceiver offer a considerable bound to the communication speed achievable on the S12X implementation. The overall computation time can be decreased by computing all cryptographic primitives on the XGATE co-processor. While the cryptographic functions are being computed on the XGATE side, the main CPU is free to execute other tasks, such as

Nonce size (bytes)	MAC size (bytes)							
	1	2	4	8	1	2	4	8
	S12X (ms)				TriCore (μs)			
1	3.680	3.740	3.860	4.115	161.4	171.4	191.4	230.5
2	3.740	3.810	3.935	4.185	172.4	182.4	205.5	241.5
4	3.870	3.935	4.055	4.295	189.4	199.5	219.5	258.5
8	4.160	4.220	4.355	4.620	225.5	236.5	256.5	295.5

Table 4.3: S12X and TriCore round trip time.

receiving messages or sending messages that have been already built. After the program implementation, the total RAM memory left for storing key chains can hold 1216 elements (16 bytes each). Having this upper limit for MEM, ℓ and σ have to be determined for best performances based on the bus speed and the wanted disclosure delay. If we consider packets of 16 bytes in size the measured bus speed for sending authenticated packets is 578 packets/second (one packet each 1.73ms) which could be considered too slow in a time critical system.

To determine the effect of the bus speed on the synchronization procedure, we measured the round-trip time for a short message exchange. The slave node that wants to start communication with the master first sends a nonce to the master. Upon receiving this request, the master sends it's response containing the MAC computed over the nonce. The measured round-trip time is the time span between the moment in which the slave sends the nonce and the moment the reception of the response from the master is over. Table 4.3 holds or measurements for different nonce and MAC sizes. For these measurements, the CAN bus speed was set to 125kbps for the S12X and 1Mbps for TriCore while HMAC-MD5 with an 8 byte key was used for computing the MAC. The length of the computed MAC is 16 bytes and we obtained the smaller MACs by truncating this original value to the desired length.

As Table 4.3 shows there is little difference in round-trip time if we change the nonce or MAC size from 1 to 8 bytes, since only one CAN frame is sent in both cases. When we change these sizes to 16 bytes the increase in round-trip time is greater due to the overhead caused by having to send two CAN frames instead of only one as the case for 1 and 8 bytes messages.

4.2.3.2 Adjusting parameters

Choosing the best combination of parameters highly depends on the application and on the devices used for implementation. We therefore tried different values for the protocol parameters on the TriCore implementation to find the best suited setup. The parameters were chosen so that they fit for authenticated communication over a time span of 10 years without the need of reinitialization. Table 4.4 contains some of the parameter combinations we tried for our TriCore implementation. The aim was to

Key size (bytes)	3 levels			3 levels			3 levels		
	$\delta(ms)$	M(bytes)	σ	$\delta(ms)$	M(bytes)	σ	$\delta(\mu s)$	M(bytes)	σ
4	10	37704	3142	1	81216	6768	315.24	120024	10002
mixed	10	75408	3142	1	162432	6768	318.08	239304	9971
16	10	150816	3142	1	324864	6768	471.44	419808	8746
Key size (bytes)	4 levels			5 levels					
	$\delta(\mu s)$	M(bytes)	σ	$\delta(\mu s)$	M(bytes)	σ			
4	329.44	15840	990	340.8	4960	248			
mixed	332.28	39520	988	343.64	13832	247			
16	479.96	57664	901	497	18400	230			

Table 4.4: Some parameters choices for the Infineon TriCore platform.

find the best parameter combinations to obtain small authentication delays (equidistant timing was used in all variants). The key chains were computed using MD5 and HMAC-MD5 was used for MACs. We underline that although MD5 does not offer collision resistance anymore it is still secure enough for our application that requires only secondary pre-image resistance. Using stronger hash functions from the SHA-2 family or from the SHA-3 candidates will impair performances without much practical justification. As computational results for these functions are available in Table 4.2, the protocol performance can be easily deduced for these cases as well. For the case of the S12X microcontrollers, due to the reduced computational power and mostly due to the bus speed reduced to 125 kbps, delays in the order of 10ms were the best we could achieve.

In the case of the Infineon TriCore, as the greatest communication overhead was caused by the CAN frame format and maximum transfer speed, we tried different key sizes. When using the whole 16 bytes of the MD5 generated key, the smallest authentication delay we could obtain was 471.44μs for 3 levels. By using smaller keys (which we obtained by truncating the 16 bytes output of MD5), the communication overhead is reduced at the cost of a lower security level. A 4 byte key used on all levels would enable an authentication delay of 315.24μs for a 3 level setup and 340.8μs for 5 levels. A better trade-off can be made by assigning different length keys to each level in order to provide enough security for the key lifetime. As an example, we assigned a 4 byte key to the first level, an 8 byte key for the second level, 12 bytes for the third level and 16 bytes for all the other levels. For a 5 level setup, each 4 byte key will have to last for 343.64μs while the 8 byte keys on the second level will have a life time of 84.88ms. We underline that these delays show the minimum achievable with our implementation and, since they are on the extreme side, reaching them is quite consuming for the computational resources of the devices as well as for the bus load. For practical settings, delays from 1 to 10ms should be easily handled by the TriCore controllers and will result in a clean deployment without consuming much of the controller's resources or communication bandwidth.

4.3 One-time signatures

Here we explore the possibility of using one-time signatures for assuring broadcast authentication at the application layer of CAN based. We find the enhanced Merkle and HORS [105] signatures to offer different trade-offs, the first is more efficient in terms of memory, while the second is more efficient in terms of signature size and verification time. Indeed, with the HORS signature we exhibit good improvements in the authentication delay. The enhanced Merkle signature also has certain advantages. More concrete, the size of the messages is quite small in most broadcast scenarios since CAN frames carry small data from sensors and actuators and this signature allows message recovery, thus small messages can be embedded in the signature. Finally, both signature schemes can be efficiently paired with time synchronization to reduce the overhead to re-initialize the public keys, which would otherwise require expensive authentication trees.

4.3.1 Signature schemes and broadcast protocol

We employ the now classical procedure of using a one-way key chain with time synchronization to commit the public keys that are used to verify the signatures. We stress that the intention is to use these public keys to authenticate the broadcast and not to assure non-repudiation. Because of this we can renounce on the classical structure of Merkle trees to authenticate them, which will be more memory and bandwidth consuming. If one wants to assure non-repudiation at some point, at the cost of extra-memory, then eventually any number of the released public keys could be signed afterwards. The signature schemes are flexible and their parameters can be used to adjust the consumed computational power, memory and bandwidth. These are discussed in detail after the protocols description.

4.3.1.1 The signature schemes

We first review the one-time signature schemes that we are going to use and give an example to underline its efficiency. In the next section we integrate this in the protocol that we are going to use. The generic principle behind both one-time signatures is to apply a simple on-way function, e.g., a hash function, over some input that plays the role of secret key and use the output as public key. However if bits are signed individually this results in an inefficient scheme, not necessarily due to the number of hash computations since these are cheap, but mainly due to the size of the signature itself, e.g., in worst case on hash for each bit. For this purpose, several improvements were proposed in the literature. The enhanced Merkle signature and HORS [105] that we discuss next employ the one-way chains in two highly distinctive fashions, a reason for which we choose to evaluate both of them in out CAN broadcast scenario.

Enhanced Merkle Signature Scheme (EMS). Given one-way function f, signature scheme EMS is a triplet of polynomial time algorithms $Gen, Sign, Ver$ where:

1. *Gen* is a probabilistic algorithm that takes as input the security level k along with two integers λ, μ and outputs the public-private key pair pk, pv, i.e., $pb = \{(f^\lambda(u_\mu), f^\lambda(v_\mu)) ,..., (f^\lambda(u_2), f^\lambda(v_2)), (f^\lambda(u_1), f^\lambda(v_1))\}$, $pv = \{(u_\mu, v_\mu), ..., (u_1, v_1)\}$ $\leftarrow Gen(1^k, \lambda, \mu)$ (here all u_i, v_i are random values of k bits each),

2. *Sign* is a deterministic algorithm that takes as input the private key pv, a message m of $\lfloor log_2(\lambda) \rfloor \cdot \mu$ bits which can be written as $m = m_\mu ... m_2 m_1$ (where each m_i has $\lfloor log_2(\lambda) \rfloor$ bits), and outputs a signature s, i.e., $s = \{(f^{\lambda - m_\mu}(u_\mu), f^{m_\mu}(v_\mu)),$ ... , $(f^{\lambda - m_2}(u_2), f^{m_2}(v_2)), (f^{\lambda - m_1}(u_1), f^{m_1}(v_1))\}$,

3. *Ver* is a deterministic algorithm that takes as input the signature and the public key and outputs message $m = m_\mu ... m_2 m_1$ if and only if $\forall i = 1..\mu, f^{\lambda - m_i}(s_i) = f^\lambda(u_i), f^{m_i}(s_i) = f^\lambda(v_i)$ or $\perp$ otherwise.

Security. The previous scheme is an improvement of the classical Merkle, but as we couldn't find proof for its security we consider to give a proof sketch here. First note that signing each component of the message is independent from another, thus it is sufficient to prove that the adversary is unable to forge any part of the signature. Let Adv be an adversary that manages to forge a signature on some message block m' with non-negligible probability ϵ_{Adv}. We use Adv to make an algorithm that inverts f with non-negligible probability on some target $y = f(x)$. The inverter chooses random $l \leq \lambda$ and random r then flips a bit b. If $b = 0$ then the inverter computes the pair $f^l(y), f^\lambda(r)$ which is set as the public key otherwise he computes and sets the public key as $f^\lambda(r), f^l(y)$. Now the adversary is allowed to make a query to the signing oracle. Let $\Pr_{\text{Bad}}$ denote the probability that Adv asks for $m_{Adv} > l$ and $b = 0$, or else $m_{Adv} < l$ and $b = 1$ which will make the oracle fail to answer and abort. Obviously $\Pr_{\text{Bad}} = 1 - l/\lambda$. Otherwise, the oracle succeeds and the adversary must output the forgery with probability ϵ_{Adv} for some $m' \neq m_{Adv}$. If $m' > m_{Adv}$ as l is also random with probability $(\lambda - l)/\lambda$ we have $m' > l$ and with probability 1/2 we have $b = 0$ which means that we can use the adversary output to invert f with probability $1/2 \cdot (\lambda - l)/\lambda \cdot \epsilon_{Adv}$. Otherwise, if $m' < m_{Adv}$ with the same probability we can invert f in the second case if $b = 1$. Summing up, the probability to invert the function is non-negligible.

HORS Signature Scheme. Given one-way function f, signature scheme HORS is a triplet of polynomial time algorithms $Gen, Sign, Ver$ where:

1. *Gen* is a probabilistic algorithm that takes as input the security parameters l, k and integers λ, μ then generates λ random k-bit values $s_1, s_2, ..., s_\lambda$ then computes $v_i = f(s_i)$ and outputs the public-private key pair pk, pv, i.e., $pb = \{\mu, f(s_1)$ $,..., f(s_\lambda)\}$, $pv = (k, s_1, ..., v_\lambda) \leftarrow Gen(1^k, l, \lambda)$,

2. *Sign* is a deterministic algorithm that takes as input the private key pv and message m then computes $h = hash(m)$ and splits h into k substrings $h_1, ..., h_\mu$ each of $\log_2(t)$ bits and outputs $s = (s_{h_1}, ..., s_{h_\mu})$ (where each h_i is interpreted as an integer index),

3. *Ver* is a deterministic algorithm that takes as input the signature s, the public key pb and message m the outputs 1 if and only if $f(s'_i) = v_i$ for each i extracted as integer index from $h(m)$.

Security. The security proofs for this scheme can be found in the original paper [105]. We mention here that the security level of this signature scheme is $\mu(\log\lambda - \log\mu - \log r)$ if the adversary obtains r signed messages of its choice.

4.3.1.2 The broadcast protocol

For each of the signature schemes we use a broadcast protocol that relies on one-way key chains. In the case of the EMS signature, the key chain is used to commit future public keys, while in the case of the HORS signature each element of the key chain forms a public key for the signature (this happens in a similar fashion to the BiBa protocol from Perrig [98]).

Time synchronization is loose and is done with synchronization error $\epsilon_{\mathcal{R},\mathcal{S}}$ which is the round-trip time of a handshake between the receiver and the sender. Usually this handshake has two steps as $1.\mathcal{R} \rightarrow \mathcal{S} : N_{\mathcal{R}}$, $2.\mathcal{S} \rightarrow \mathcal{R} : Sig_{\mathcal{S}}(t^{\mathcal{S}}_{sync}, N_{\mathcal{R}})$ where t_{sync} denotes time on the sender side and $N_{\mathcal{R}}$ is a nonce chosen by the receiver. However, as we need to keep the synchronization error as small as possible, we will not use a digital signature and instead we will use a *MAC* which is several order of magnitudes cheaper. By using it, the synchronization error gets to the order of several milliseconds, which is accurate enough for high speeds of the broadcast. Afterwards, the receiver $\mathcal{R}$ can estimate at any point $t_{current}$ that the time on the sender's side $\mathcal{S}$ is $\mathcal{T}_{\mathcal{S},\mathcal{R}}(t_{current}) \in [t^{\mathcal{S}}_{sync} + t_{current} - t^{\mathcal{R}}_{sync}, t^{\mathcal{S}}_{sync} + t_{current} - t^{\mathcal{R}}_{sync} + \epsilon_{\mathcal{R},\mathcal{S}}]$.

Broadcast with EMS. Given signature scheme EMS and the roles sender $\mathcal{S}$ and receiver $\mathcal{R}$ we define protocol Broadcast-EMS$_{\mathcal{S},\mathcal{R}}[\lambda, \mu, \delta]$ as the following actions performed by $\mathcal{S}$:

1. *Initialization*: $\mathcal{S}$ generates a key chain by using a random $\mathcal{K}_0$ and computing $\mathcal{K}_n = f(\mathcal{K}_{n-1}), \forall i = 1..n$, then he commits the tip of its top level-chain, i.e., $\mathcal{K}_n$, the disclosure delay δ and the public key obtained by running $Gen(1^k, \lambda, \mu)$,

2. *Commitment*: $\mathcal{S}$ sends at any point in time interval $[t_{start} + i\cdot\delta, t_{start} + (i+1)\cdot\delta - \xi]$ (here ξ is a tolerance value to prevent the sender to commit a *MAC* to close to the disclosure point which will increase the chance for a receiver to drop the packet) a fresh public key pb generated by using $Gen(1^k, \lambda, \mu)$ and a *MAC* computed with $\mathcal{K}_{i+1}$ on it, i.e, $MAC_{\mathcal{K}_{i+1}}(pb)$,

3. *Key Disclosure*: $\mathcal{S}$ sends at time $t_{start} + i\cdot\delta$ the corresponding key from the key chain, i.e., $\mathcal{K}_i$,

4. *Authentic Broadcast*: $\mathcal{S}$ sends at any time in $[t_{start} + i\cdot\delta, t_{start} + (i+1)\cdot\delta - \xi]$ message m as a signature with message recovery $s = Sign(m, pv_{last})$ (here pv_{last} is the most recently generated private key);

and $\mathcal{R}$ respectively:

1. *Initialization*: $\mathcal{R}$ receives the initialization information of the sender, i.e., $\mathcal{K}_n$, the disclosure interval δ and the public key pk,

2. *Time Synchronization*: $\mathcal{R}$ performs a loose time synchronization with $\mathcal{S}$, such that the synchronization error $\epsilon_{\mathcal{R},\mathcal{S}} << \delta$,

3. *Receive Keys and Commitments*: $\mathcal{R}$ receives $\mathcal{K}_i$ and checks if $f(\mathcal{K}_i) = \mathcal{K}_{i-1}$ and discards it otherwise. Any *MAC* computed with $\mathcal{K}_i$ that is received after $\mathcal{T}_{\mathcal{S},\mathcal{R}}(i \cdot \delta)$ is discarded. Any public key for which there exists a valid *MAC* and key $\mathcal{K}$ that can verify it is deemed authentic,

4. *Message Verification*: $\mathcal{R}$ runs $Ver(pk_i, sig_i)$ for any valid public key and deems authentic any output different from $\perp$.

Security. The signature scheme was proved to be secure while the security of such protocols based on time synchronization is well known. The informal argument is that from $MAC_{\mathcal{K}}(M)$ and $f(\mathcal{K})$ an adversary cannot produce $MAC_{\mathcal{K}}(M')$ for any $M' \neq M$ since $\mathcal{K}$ is not known as well as it cannot be found from $f(\mathcal{K})$. By the time $\mathcal{K}$ is released it is already too late for the adversary to send a *MAC* and a message as they will not be anymore accepted by the receiver due to the time constraint. For completeness we can give a more formal proof sketch here. It is commonly acknowledged that although random oracles do not give an absolute proof they can be used at least as a sanity check to prove the security of protocols. If we assume that the oracle $\mathcal{O}^f$ that computes function f can be replace by a random oracle $\mathcal{O}^R$, which outputs k bits, the proof is straight forward. Assume that the adversary has witness polynomially many queries $p(k)$ to oracle $\mathcal{O}^R$. Suppose at some point the adversary is forced to produce $MAC_{\mathcal{K}}(M_{Adv})$ for some message of its choice. But the adversary knows just $\mathcal{O}^R(\mathcal{K})$ which is the output of the random oracle and $\mathcal{K}$ is unpredictable subject to the fact that it may have been already asked by the adversary to $\mathcal{O}^R$. This means he can guess it and produce a valid *MAC* only with probability $1/(2^k - p(k))$ - which is negligible.

Broadcast with HORS. Given signature scheme HORS and the roles sender $\mathcal{S}$ and receiver $\mathcal{R}$ we define protocol Broadcast-HORS$_{\mathcal{S},\mathcal{R}}[\lambda, \mu, \delta]$ as the set of following actions performed by $\mathcal{S}$:

1. *Initialization*: $\mathcal{S}$ generates a key chain starting from random $\mathcal{K}_0, ..., \mathcal{K}_\lambda$ and computing $\mathcal{K}_j^i = f(\mathcal{K}_i^{j-1}), \forall i = 1..\lambda, j = 1..\mu$, then he commits the tip each chain, i.e., $\mathcal{K}_i^\mu$,

2. *Authentic Broadcast*: $\mathcal{S}$ sends at any time in $[t_{start} + i \cdot \delta, t_{start} + (i+1) \cdot \delta - \xi]$ message m along with its signature computed with HORS having as secret key input the keys from the current disclosure interval $\mathcal{K}_0^i, \mathcal{K}_1^i, ..., \mathcal{K}_\lambda^i$ (the number of messages signed in each time interval depends on the security level and signature parameters);

3. *Key Disclosure*: $\mathcal{S}$ sends at time $t_{start} + i \cdot \delta$ all the keys from the current interval that were not disclosed as HORS signature (to save some bandwidth, sending these keys can be skipped since the receivers can validate future signatures with previously received keys, but note that this will increase verification time on receivers)

and $\mathcal{R}$ respectively:

1. *Initialization*: $\mathcal{R}$ receives the initialization information of the sender, i.e., $\mathcal{K}_i^{\mu}, \forall i = 1..\lambda$ and the disclosure interval δ,

2. *Time Synchronization*: $\mathcal{R}$ performs a loose time synchronization with $\mathcal{S}$, such that the synchronization error $\epsilon_{\mathcal{R},\mathcal{S}} << \delta$,

3. *Receive Keys and Commitments*: $\mathcal{R}$ receives keys $\mathcal{K}_i^j$ and checks if $f(\mathcal{K}_i^{j-1}) = \mathcal{K}_i^j$ and discards it otherwise.

4. *Message Verification*: $\mathcal{R}$ runs $Ver(pk_i, sig_i)$ for any signature that is received in the correct time interval and deems authentic any input that is correctly verified.

Security. The security can proved by simply constructing a forger for the HORS signature. In this case a challenger can simply use the public key of the signature to be forged to build key chains and further simulate the broadcast protocol with the hope that an adversary will forge the target signature.

4.3.1.3 Efficiency

We start by analyzing various trade-offs that can be achieved with the enhanced Merkle signature then we compare it to RSA signatures and finally to HORS. The main conclusion is that in general HORS would be more efficient in terms of verification speed and bus load (while it is less efficient in terms of memory requirements) and in the experimental section we provide the computational and communication bounds that we reached for HORS.

Enhanced Merkle Signature. To judge efficiency it is relevant to consider the number of bits that can be signed with a committed public key. Since the length of the chains, i.e., λ, and their number, i.e., μ, is bounded by the computational power in time δ we could write $\lambda = \sigma \cdot \delta / \mu$, where σ denotes computational speed, i.e., the number of function computations per second. Having a fixed σ it is relevant to decide which will be more efficient from a computational point of view: to have larger μ and shorter λ or vice-versa.

Figure 4.6 depicts the variation of signed bits and signature size with parameter μ of the signature scheme. The plot is depicted for a speed fixed to 2000 one-way function computations per second, which is around the average of our experimental results presented in the next section. As can be seen, larger μ means more bits can be signed, but require much more bandwidth.

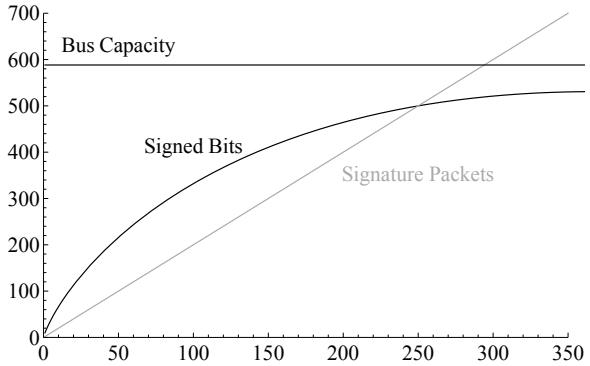


Figure 4.6: Variation of signed bits and signature size with $\mu \in [0, 350]$

Figure 4.7 shows the variation of the number of signed bits with computational speed σ and μ. Clearly computational speed cannot compensate enough the decrease in the chain length as it results in division with a logarithmic factor. However, decreasing the chain length results in the same expense of bandwidth as can be seen from figure 4.6. Thus, higher computational speeds certainly help up to some point when one needs to decrease the length λ in order to allow an increase in μ.

Figure 4.8 shows the variation of the number of signed bits with computational speed σ and busload B. The plot is given for $\mu \in [0, 500]$ and $B \in [0, 1]$. The maximum number of signed bits is achieved by taking μ equal to half the maximum number of packets that can be send on the bus and then computing λ according to the computational speed and the disclosure delay which is fixed to 1s in this plot. As can be seen, the main limitation for the number of signed bits is given by the bus speed. For example in a fault tolerant CAN with 128kbps, the number of signed bits will not exceed 1.2kbps even if a hash computation does not exceed $100\mu s$. If the bus speed is increased to 1 Mbps, as in high speed CAN, then the number of signed bits can get up to 2.5 kbps. These results hold for the EMS signature, for HORS we discuss the performance in the experimental results section.

Comparison with RSA. A relevant thing about this signature in the way it was presented before is that it allows message recovery. In an environment with constrained bandwidth this becomes relevant with respect to performance. We consider to outline the efficiency of this scheme by a short comparative example with an RSA based signature. If a k bit message is to be signed with fixed λ then $2 \cdot \lambda \cdot \lceil k/log_2\lambda \rceil$ one-way function computations are needed and the signature size is $2 \cdot \lceil k/log_2\lambda \rceil$. Now consider an 1024 bit RSA compared to an MD5. Indeed these primitives are not very strong for today requirements but the proportion gets even worse for the RSA as bigger public keys are used. By taking timings from OpenSSL, on a notebook with an Intel Core2Duo processor at 1.4 Ghz, we get that MD5 is 3670 times faster than the RSA private key operation which is done at signing. Now to sign a 128 bit message assuming $\lambda = 64$ we can process 8 bits at a time which results in $2 \cdot 64 \cdot 16 = 2048$

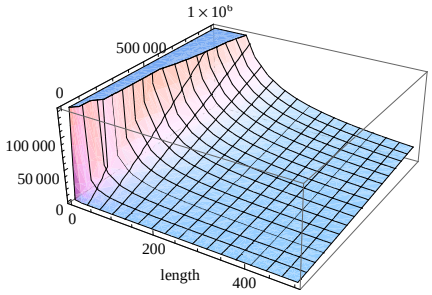


Figure 4.7: Variation of signed bits with $\sigma \in [0, 1 \times 10^6]$ and $\mu \in [0, 500]$

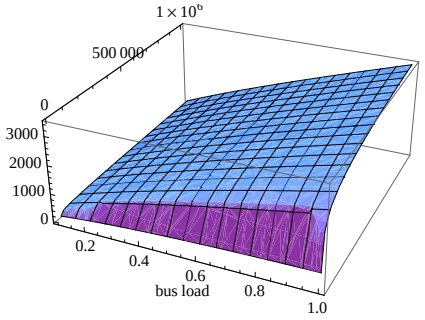


Figure 4.8: Variation of signed bits with busload (from 0 to 100%) and $\sigma \in [0, 1 \times 10^5]$

computations of MD5. That is still almost half the time required by RSA. But indeed it results in a signature that is $2 \cdot 16 \cdot 128 = 4096$ bit long, which is 4 times larger than for RSA. But remember that in our scenario short messages are more common and consider a message is 32 bits. With length $\lambda = 64$ again 8 bits can be processed at a time and we get a signature of the same size as the RSA, i.e., $2\dot{4}\cdot 128 = 1024$, but only $2 \cdot 4 \cdot 64 = 512$ MD5 computations are needed which is 7 times faster than the RSA. For shorter messages, this improves even more, a more detailed analysis is done after the complete description of the protocol. It is commonly known that one can improve on this even more by signing the bits of m only by using the $f^i(u)$ values from the above signature scheme and using the $f^j(v)$ values to sign the sum $\sum_{i=1,\mu}(2^\lambda - m_i)$. In the worst case this will require the same computational costs and size for the signature while in the best case it requires only half the size. To keep the following description simple we leave this just as potential improvement in a practical application.

Comparison with HORS. If we assume the message to be signed is k bits then having length λ for the chains in the enhanced Merkle signatures and HORS then the following constraints hold. The size of the signature, which gives the bus load, is $2k/\log_2 \lambda$ in case of EMS and twice as short in case of HORS, i.e., $k/\log_2 \lambda$. However, in terms of memory HORS requires λ key chains to be stored, while EMS requires only $k/\log_2 \lambda$ key chains which is obviously less. In terms of verification speed HORS is again superior to EMS requiring only $k/\log_2 \lambda$ as opposed to $\lambda k/\log_2 \lambda$ required by EMS. In the experimental section we give practical data on the efficiency of both these schemes.

4.3.1.4 Further improvements: recycling unused keys

Recycling Public keys that were authenticated but unused can be safely used in forthcoming time intervals. The only restriction that must be taken into account is to avoid memory exhaustion attack on the receiver. This is because and adversary may intentionally break the communication channel between $\mathcal{R}$ and $\mathcal{S}$ which will determine the sender to further store public keys until its resources will exhaust. To avoid these a

Hash function	Execution time			
	@ 40MHz		@ 80MHz	
	S12X	XGATE	S12X	XGATE
MD5	738μs	316μs	367.5μs	156.8μs
SHA1	2285μs	1000μs	1144μs	501μs
SHA-256	5490μs	3145μs	2740μs	1572μs

Table 4.5: Performance of S12X and XGATE in computing hashes.

maximum life-span of the public keys can be fixed.

It may be also tempting to recycle unused parts of the chains corresponding to the public keys. If the new tips are authenticated this can be done but combining this with the previous procedures results in an unsafe protocol. For example consider that $\mathcal{S}$ decides to use an unused part of a public key and authenticates it using the top level chain. Now an adversary that has intentionally broken the communication between $\mathcal{S}$ and $\mathcal{R}$ can use the newly committed tips to forge a signature. Because of this, reusing parts of the public keys should be avoided.

4.3.2 Experimental results

In order to confirm our theoretical results we made some tests on the Freescale S12X microcontroller. For an improvement of the overall performance of the implemented protocols we took advantage of the XGATE co-processor to compute cryptographic primitives. Additionally we observed that the operating bus frequency of the microcontroller can be pushed beyond the 40 MHz limit stated by the datasheet without affecting its functionality. We were able to successfully use a maximum frequency of 80 MHz at which the microcontroller was stable.

4.3.2.1 Computational performance

Using the main S12X CPU for computing cryptographic primitives, communication and all other necessary operations would lead to a poor performance even if the microcontroller is overclocked at 80 MHz. To compensate for the small frequency we had to reduce the load of the main CPU and we did this by using the XGATE co-processor for executing all cryptographic computations. When a hash for example has to be executed, the main S12X CPU has to issue a software request to the XGATE co-processors. Until the computation is done on XGATE, the main CPU will be free to execute other tasks.

Three well known hash functions have been implemented: MD5, SHA1 and SHA-256. The execution speed for each of these functions was tested both on the S12X CPU and XGATE. The results are presented in Table 4.5 for the case of using the maximum documented frequency and the overclocked one. The input for each hash function was equal in length to each specific hash output.

As the overall authentication overhead is also affected by the commitment of the public keys we also evaluate the time needed to perform a MAC on S12X. We used

Input length	Execution time			
	@ 40MHz		@ 80MHz	
	S12X	XGATE	S12X	XGATE
HMAC-MD5				
64	5.39*ms*	2.310*ms*	2.695*ms*	1.154*ms*
128	6.02*ms*	2.580*ms*	3.010*ms*	1.288*ms*
256	7.27*ms*	3.110*ms*	3.635*ms*	1.558*ms*
512	9.78*ms*	4.185*ms*	4.890*ms*	2.090*ms*
1024	14.78*ms*	6.33*ms*	7.39*ms*	3.165*ms*
HMAC-SHA1				
64	17.78*ms*	7.79*ms*	8.89*ms*	3.895*ms*
128	19.95*ms*	8.74*ms*	9.96*ms*	4.375*ms*
256	24.25*ms*	10.64*ms*	12.14*ms*	5.32*ms*
512	32.95*ms*	14.44*ms*	16.48*ms*	7.22*ms*
1024	50.4*ms*	22.05*ms*	25.15*ms*	11.04*ms*
HMAC-SHA256				
64	43.1*ms*	24.80*ms*	21.55*ms*	12.42*ms*
128	48.4*ms*	27.90*ms*	24.20*ms*	13.94*ms*
256	59.0*ms*	34.00*ms*	29.55*ms*	17.00*ms*
512	80.1*ms*	46.2*ms*	40.05*ms*	23.15*ms*
1024	122.4*ms*	70.7*ms*	61.20*ms*	35.35*ms*

Table 4.6: Performance of S12X and XGATE in computing MACs.

HMAC together with the three hash functions presented above and a 128 byte key. Table 4.6 holds the execution time we obtained for different message sizes.

4.3.2.2 Protocol performance

As shown in the previous section (Table 4.5), the computation of one MD5 is done in 156.8μs on XGATE at a frequency of 80 MHz. With this speed, considering data blocks of 64 bits with $\lambda = 64$ and $\mu = 47$ (bounded by the computational speed) we get a bus load of around 16% and approximately 286 bits can be authenticated in one second. For a bus load of 50%, having $\lambda = 21$ and $\mu = 147$, the authentication speed increases to 652 bps. To reach an authentication speed of 1000 bps we can use $\lambda = 10$ and $\mu = 294$ but at the cost of a bus load of 100%. This may not seem much, but it allows a flexible tradeoff between the length and the number of signed messages. For example, in the first case, at a bus load of only 16% a number of 47 messages of 8 bits can be signed in each second which is enough to hold critical data from analog-to-digital converters, etc., while 74% of the bus is free and can be used for other non-critical tasks. This amount of messages cannot be signed with a public key primitive such as the RSA, which requires hundreds of milliseconds on S12. This contrast shows the efficiency of the employed mechanism.

For implementing the HORS protocol we looked for a suitable way of adapting it for devices with lower computational powers. We adapted our setup by adding a master node which has the sole purpose of authenticating messages that are broadcasted on the CAN bus. To allow this, each of the other participants to the communication (which

will be called slave) has to share a secret key with the master. In this way, when a slave wishes to send an authenticated message, it will put one frame on the bus containing the message and a counter, followed by another frame which will contain a MAC computed using the preshared key over the message-counter pair. The master checks the authenticity of the message using the key associated with the sender ID and if it succeeds it will send the HORS authentication information to all nodes. We chose to use a master node because HORS involves using a high number of key chains which would need a considerable amount of memory for storing. Since the bus nodes are constrained even in what regards the available memory, the most cost effective solution would be to have only one device with higher computational speed and storage that plays the role of a master.

Even in this master-slave setup it is not an easy task to find a suitable parameter combination that will consume as little as possible from the slave constrained resources and in the same time allow for a good communication speed. We tested a setup that uses $t = 1024$ key chains of $k = 48$ bits each as the signature chain and MD5 as our f function. The result of the MD5 is split into $k = 7$ substrings of 10 bit each ($\log_2(t)$). As the master node, we used a laptop (Intel Core2Duo CPU T7700@2.4GHz) along with a CANcardXL (PCMCIA device) and CANcab 1054mag to enable the PC to communicate over CAN. Using these parameters in the master-slave setup described above we were able to reach an authentication delay of 30ms when setting the S12X chip frequency to 40MHz without employing the XGATE coprocessor. By increasing the frequency to 80MHz and using XGATE to perform the cryptographic computations this delay can be decreased to around 15ms. Alternatively, this leads to 30–60 authentic packets each second while the size of each packet is not bounded by any parameters (different to the case of the EMS scheme) except for bus speed.

4.4 LiBrA-CAN

Our third approach to providing authentication at the application layer is LiBrA (which stands for Lightweight Broadcast Authentication). We begin with a description of the frame structure employed in our protocol. Then we outline the main authentication scheme which builds upon keys shared between groups of receivers, a procedure which we call key splitting. Further, we discuss some variations of the main scheme that can be used for different trade-offs. Subsequently we introduce a construction which we call Linearly Mixed MAC (*LM-MAC*) which gives additional security benefits.

4.4.1 Protocol description

4.4.1.1 Frame structure

As a general procedure, we separate between frames that carry messages and frames that carry authentication tags. This seems to be a correct option due to a widely employed CAN mechanism which is ID filtering that is used to restrict certain frames to

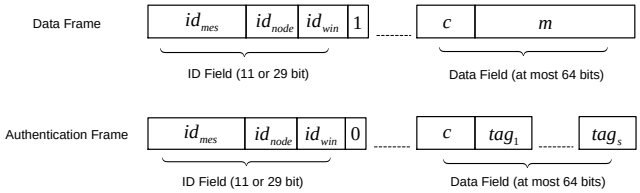


Figure 4.9: Data frames and authentication frames

arrive to a particular node. While we do want to keep this feature, we want the node to be able to carry additional authentication tasks, e.g., in the case of the two-stage authentication discussed further, a reason for which we must assure that the authentication frames are able to reach the node and thus they may need to have a different ID than the message frame. The last bit of the identifier field specifies whether a frame carries an authentication tag or message, a procedure that is employed in our experimental setup.

Larger data blocks or authentication tags can be split across multiple frames with the same ID field and counter. Other adjustments can be done at the implementation level. For example, since the ID field is quite short, both the node and window identifiers (which denote the source and the number of the authentication frame) can be moved in the data field. We preferred to place these identifiers in the ID field since it is a frequent choice of developers to place a unique ID for each node in the CAN ID field. But indeed, such an option can affect real-time requirements and for this purpose placing these IDs in the data field is safer. The size of the counter c could be roughly around 20–40 bits but this greatly depends on the bus speed (which determines the number of packets released each second).

4.4.1.2 The main scheme: centralized authentication

A master oriented communication makes sense since it is practical to have one node with higher computational power that can take care of the most intensive part of the authentication. Figure 4.10 shows the master node and the slave nodes connected to the bus, it also outlines the keys that are shared between nodes. For the key sharing procedure, all slaves register to the master which distributes the keys. In practice associating nodes to a group and sharing the keys is done by standard techniques, e.g., key-exchange protocols, we do not insist on this since such issues are straightforward to solve.

In the main scheme we make use of Mixed Message Authentication Codes (*M-MAC*) which amalgamate more MACs into one. Here we give an abstract definition for this construction while in a forthcoming section we provide a more elaborate instance with additional security properties. Indeed, the easiest way to build an *M-MAC* is simply

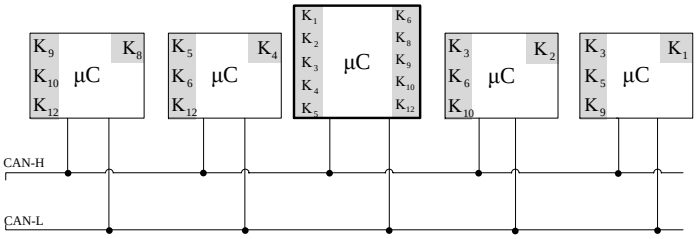


Figure 4.10: Master and slave microcontrollers (μC) in a setting for centralized authentication

by concatenating multiple tags, such a construction is fine for our protocol and can be safely embodied in the main scheme (still, we can achieve more security with the *LM-MAC* introduced in an upcoming section).

Construction 1. *(Mixed Message Authentication Code) A mixed message authentication code M-MAC is a tuple* (Gen, Tag, Ver) *of probabilistic polynomial-time algorithms such that:*

1. $\mathbb{K} \leftarrow Gen(1^{\ell}, s)$ *is the key generation algorithm which takes as input the security parameter* ℓ *and set size s then outputs a key set* $\mathbb{K} = \{k_0, k_1, ..., k_s\}$ *of s keys,*
2. $\tau \leftarrow Tag(\mathbb{K}, \mathbb{M})$ *is the MAC generation algorithm which takes as input the key set* $\mathbb{K}$ *and message tuple* $\mathbb{M} = (m_0, m_1, ..., m_s)$ *where each* $m_i \in \{0,1\}^*$ *then outputs a tag* τ *(whenever needed, to avoid ambiguities on the message and key, we use the notation* $M\text{-}MAC_{\mathbb{K}}(\mathbb{M})$ *to depict this tag),*
3. $v \leftarrow Ver(k, m, \tau)$ *is the verification algorithm which takes as input a key* $k \in \mathbb{K}$*, a message* $m \in \{0,1\}^*$ *and a tag* τ *and outputs a bit v which is* 1 *if and only if the tag is valid with respect to the key k, otherwise the bit v is* 0*. For correctness we require that if* $k \in \mathbb{K}$ *and* $m \in \mathbb{M}$ *then* $1 \leftarrow Ver(k, m, Tag(\mathbb{K}, \mathbb{M}))$*.*

The centralized scheme is summarized by the next construction. For simplicity of the exposition, since the main scheme is used to authenticate the same message to all nodes (rather than authenticate a tuple of messages as in the cumulative authentication scheme), we replace $\mathbb{M}$ with a simple array that points out the values that are authenticated, e.g., $id_{node}, id_{win}, c, m$, etc. Obviously in this case the *M-MAC* receives as input a message tuple of *s* identical messages.

Construction 2. *(Centralized Authentication) Given a mixed message authentication code algorithm M-MAC for some security parameter* ℓ*, size s and a group of n nodes, we define protocol* $CN\text{-}CAN\text{-}LiBrA_{\mathcal{M},\mathcal{S}^*}(M\text{-}MAC, \ell, s, n, b, w)$ *as the following set of actions for the master* $\mathcal{M}$*:*

1.$Setup(\ell, n, s)$ *on which master* $\mathcal{M}$ *generates all subsets of s slaves out of n slaves, let* $t = \binom{n}{s}$ *be the number of subsets, and randomly picks t keys, each of* ℓ *bits, then*

places them in the keyset $\mathbb{K}_{\mathcal{M}} = \{k_1, k_2, ..., k_t\}$. Subsequently master $\mathcal{M}$ uses a secure channel to send each node the corresponding keys (alternatively these keys can be distributed in an off-line manner). Let $\mathbb{K}^i_{\mathcal{S}} = \{k_1, k_2, ..., k_{t'}\}$ with $t' = \binom{n-1}{s-1}$ denote the key set received by each slave $\mathcal{S}$.
2.RecMes$(id_{node}, id_{win}, c, m)$ on which master $\mathcal{M}$ receives a data frame containing message m from slave $\mathcal{S}$ checks if the counter is up-to-date then stores the packet in a queue of messages to be authenticated.
3.RecTag$(id_{node}, id_{win}, c,$ M-MAC$_{\mathbb{K}^i_{\mathcal{S}}}(id_{node}, id_{win}, c, m))$ on which master $\mathcal{M}$ receives an authentication frame containing tag M-MAC$_{\mathbb{K}^i_{\mathcal{S}}}(id_{node}, id_{win}, c, m)$ from slave $\mathcal{S}$. Further, the master retrieves the packet matching the identifiers and counter from the queue and if the message proves to be authentic, i.e., $1 \leftarrow Ver((id_{node}, id_{win}, c, m), k,$ M-MAC$_{\mathbb{K}^i_{\mathcal{S}}}(id_{node}, id_{win}, c, m))$, $\forall k \in \mathbb{K}^i_{\mathcal{S}}$, he proceeds to authenticating the tag to other nodes with SendTag$(id_{node}, id_{win}, m, \mathbb{K}^i)$. If the message is not available in the queue then the tag is discarded (subsequently an error message can be sent).
4.SendTag$(id_{node}, id_{win}, m, \mathbb{K}^i)$ on which master $\mathcal{M}$ after receiving a message and its valid tag, groups all the remaining keys $\mathbb{K}_{\mathcal{M}} \setminus \mathbb{K}^i_{\mathcal{S}}$ in sets of size v then for each such set $\tilde{\mathbb{K}}^j_{\mathcal{S}}$ computes M-MAC$_{\tilde{\mathbb{K}}^j_{\mathcal{S}}}(id_{node}, j, m)$ and broadcasts it in authentication frames with node identifier id_{node} and window identifier set to j (obviously there are $|\mathbb{K}_{\mathcal{M}} \setminus \mathbb{K}^i_{\mathcal{S}}|/v$ windows).

and for each of the slaves $\mathcal{S}$:*

1.Setup(ℓ, n, s) on which slave $\mathcal{S}_i$ obtains its key set $\mathbb{K}^i_{\mathcal{S}} = \{k_1, k_2, ..., k_{t'}\}$ with $t' = \binom{n-1}{s-1}$ from master $\mathcal{M}$ (either offline or via a secure channel).
2.RecMes$(id_{node}, id_{win}, c, m)$ on which slave $\mathcal{S}$ receives a data frame containing message m from another slave $\mathcal{S}_j$ and proceeds similarly to master $\mathcal{M}$ by storing it in a queue of messages to be authenticated.
3.RecTag$(id_{node}, id_{win}, c,$ M-MAC$_{\mathbb{K}^j_{\mathcal{S}}}(id_{node}, id_{win}, c, m))$ on which $\mathcal{S}_i$ receives an authentication frame containing tag M-MAC$_{\mathbb{K}^i_{\mathcal{S}}}(id_{node}, id_{win}, c, m))$ from the master $\mathcal{M}$ or another slave $\mathcal{S}_j$ and verifies for all keys $k \in \mathbb{K}^i \cap \mathbb{K}^j$ if the tag is correct. If for all keys in its keyset a correct tag was received
then message m is deemed authentic.
4.SendMes$(m, \mathbb{K}_i)$ on which slave $\mathcal{S}_i$ whenever wants to broadcast a message m increments its local counter, computes the tag M-MAC$_{\mathbb{K}^i_{\mathcal{S}}}(id_{node}, 0, c, m)$ with its keyset $\mathbb{K}^i$ and sends the data frame containing m and an authentication frame containing the tag on the bus (note that in the case of slaves id_{win} is set to 0).

Example 1. The key allocation done by the *Setup* procedure allows the keys to be split between groups of *n* slaves. Here we clarify our intentions with the *key splitting* procedure by giving an example. Table 4.7 shows the groups that can be formed in the case of 4 nodes. If we consider groups formed by exactly 2 nodes we have $\binom{4}{2} = 6$ groups and each two nodes share exactly $\binom{2}{0} = 1$ group. Table 4.7 outlines the groups shared by $\mathcal{S}_1$, i.e., G_9, G_{10}, G_{12}, and those shared by $\mathcal{S}_2$, i.e., G_5, G_6, G_{12}. Note that they intersect in one group G_{12}. In Table 4.8 the case of $n = 4$ and $n = 8$ nodes are

	G_0	G_1	G_2	G_3	G_4	G_5	G_6	G_7	G_8	G_9	G_{10}	G_{11}	G_{12}	G_{13}	G_{14}	G_{15}
$\mathcal{S}_1$	0	0	0	0	0	0	0	0	1	(1)	(1)	1	(1)	1	1	1
$\mathcal{S}_2$	0	0	0	0	1	[1]	[1]	1	0	0	0	0	[1]	1	1	1
$\mathcal{S}_3$	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
$\mathcal{S}_4$	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1

Table 4.7: Possible groups with 4 nodes, groups of size 2 outlined in gray

				Authentication bits from one *M-MAC* (%)							
n	k	groups	sub-groups	$l=0$	$l=1$	$l=2$	$l=3$	$l=4$	$l=5$	$l=6$	$l=7$
4	1	4	1	25	25	25	25	-	-	-	-
4	2	6	3	50	33	33	16	-	-	-	-
4	3	3		75	25	0	0	-	-	-	-
8	1	8	1	12.5	12.5	12.5	12.5	12.5	12.5	12.5	12.5
8	2	28	7	25	21	17	14	10	7	3.5	0
8	3	56	21	37.5	26	17	10	5	1.7	0	0
8	4	70	35	50	28	14	5	1.4	0	0	0
8	5	56	35	62	26	8.9	1.7	0	0	0	0
8	6	28	21	75	21	3.5	0	0	0	0	0
8	7	8	7	87	12.5	0	0	0	0	0	0

Table 4.8: Authentication rate in the case of $n = 4, 8$ participants, groups of size k and l corrupted nodes

explored, with complete groups of all sizes k and any number of corrupted nodes l. The total number of groups and the subgroup shared by each node as well as the percentage of secure bits, i.e., bits that cannot be forged by an adversary, from each *M-MAC* are outlined. Indeed, the percent of authenticated bits from each tag is higher and decreases significantly with the number of corrupted nodes.

4.4.1.3 Variations of the main scheme: two-stage and cumulative authentication

For practical reasons we discuss two variations of the main scheme. In the experimental results section, the first variation is shown to have certain advantages in front of the main scheme for scenarios when nodes have equal computational power.

In the case of *two-stage authentication* we assume a scenario in which only slave nodes are present, i.e., nodes with equal computational power. In this case each node can start broadcasting by sending a tag which includes only a part of the keys for the subgroups that he is part of and a second slave (pointed out by some flag, or predefined in protocol actions) continues with the authentication. The procedure is repeated until the desired number of authentication frames is reached. Various ways for tag allocation can be imagined. Consider the case of 8 nodes in subgroups of size 3 and 4 authentication frames (codenamed *TS-8S3F4*). If *M-MAC*s are used then these can be set up to work in $GF(2^{16})$ or $GF(2^{32})$. Subsequently each node sends

an *M-MAC* with keys for 4 of the nodes (or 2 in case $GF(2^{32})$) and the nodes reply in a round-robin fashion (note that a frame carries at most 64 bits). To save some computational power and have even more flexibility in tag allocation it is also possible to skip the use of the *M-MAC*. In Table 4.9 we give an example for this case. Each row corresponds to one of the 8 slaves and each column to one of the 56 groups that are formed with 3 slaves, × is used as placeholder to denote that a node is part of a group. Here f_j^i denotes the j-th part of frame i and the authentication is started by slave $\mathcal{S}_1$ with frame f_*^1 followed by $\mathcal{S}_2$ with f_*^2 then again $\mathcal{S}_1$ with f_*^3 but this time followed by $\mathcal{S}_3$ with f_*^4 (here $*$ is a placeholder for any of the frame components). We can set the size of each tag in f_*^2 and f_*^2 to 16 bits and for f_*^1 and f_*^3 use around 5-7 bits for each tag. This will result in a security level of around 64 bits for each node.

	G_1	G_2	G_3	G_4	G_5	G_6	G_7	G_8	G_9	G_{10}	G_{11}	G_{12}	G_{13}	G_{14}	G_{15}	G_{16}	G_{17}	G_{18}	G_{19}	G_{20}	G_{21}	G_{22}	G_{23}	G_{24}	G_{25}	G_{26}	G_{27}	G_{28}
$\mathcal{S}_1$	f_1^1	f_2^1	f_3^1	f_4^1	f_5^1	f_6^1	f_7^1	f_8^1	f_9^1	f_{10}^1	f_1^3	f_2^3	f_3^3	f_4^3	f_5^3	f_6^3	f_7^3	f_8^3	f_9^3	f_{10}^3	f_{11}^3							
$\mathcal{S}_2$	×	×	×	×	×	×																f_1^2	f_2^2	f_3^2	f_4^2	×	×	×
$\mathcal{S}_3$	×						×	×	×	×	×											×	×	×	×	×		
$\mathcal{S}_4$		×					×					×	×	×	×							×					×	×
$\mathcal{S}_5$			×					×				×				×	×	×					×				×	
$\mathcal{S}_6$				×					×				×			×			×	×				×				×
$\mathcal{S}_7$					×					×				×			×		×		×				×			
$\mathcal{S}_8$						×					×				×			×		×	×					×		

	G_{29}	G_{30}	G_{31}	G_{32}	G_{33}	G_{34}	G_{35}	G_{36}	G_{37}	G_{38}	G_{39}	G_{40}	G_{41}	G_{42}	G_{43}	G_{44}	G_{45}	G_{46}	G_{47}	G_{48}	G_{49}	G_{50}	G_{51}	G_{52}	G_{53}	G_{54}	G_{55}	G_{56}
$\mathcal{S}_1$																												
$\mathcal{S}_2$	×	×	×	×	×	×	×	×																				
$\mathcal{S}_3$									f_1^4	f_2^4	f_3^4	f_4^4	×	×	×	×	×	×										
$\mathcal{S}_4$	×	×							×	×	×	×							×	×	×	×	×	×				
$\mathcal{S}_5$			×	×	×				×				×	×	×				×	×	×				×	×	×	
$\mathcal{S}_6$			×			×	×			×			×			×	×		×			×	×		×	×		×
$\mathcal{S}_7$	×			×		×		×			×			×		×		×		×		×		×	×		×	×
$\mathcal{S}_8$		×			×		×	×				×			×		×	×		×	×		×	×		×	×	×

Table 4.9: Example of tag scheduling with two-stage authentication *TS-8S2F4* (8 nodes with groups of size 3)

Since in some scenarios small delays may be acceptable, we can take benefit of them and increase the efficiency of the main scheme. In the *cumulative authentication* scheme a timer can be used and all messages are accumulated by the master over a predefined period δ then authenticated at once (this procedure can be employed in the slave-only settings as well). While this introduces an additional delay δ, similar to the case of the TESLA protocol, this delay can be chosen as small as needed to cover application requirements. Different to the case of the delay from TESLA like protocols, this delay is not strongly constrained by external parameters (such as oscillator precision, synchronization error, bus speed, etc.).

4.4.1.4 Increasing security with *LM-MAC*s (Linearly Mixed MACs)

As outlined in our abstract description, *M-MAC*s use an array of keys to build a tag which is verifiable by any of the keys. The first security property which we require for an *M-MAC* is *unforgeability* and is a standard property for any MAC code, thus it merely derives from the main building block. We do develop on this by requiring a new property which we call *strong non-malleability* and which we show to be achievable by

our more advanced *LM-MAC* construction.

Construction 3. *(Linearly Mixed MAC) We define the LM-MAC as the tuple of probabilistic polynomial-time algorithms* (Gen, Tag, Ver) *that work as follow:*

1. $\mathbb{K} \leftarrow Gen(1^{\ell}, s)$ *is the key generation algorithm which flips coins and returns a key set* $\mathbb{K} = \{k_0, k_1, ..., k_s\}$ *where each key has* ℓ *bits (*ℓ *is the security parameter of the scheme),*

2. $\tau \leftarrow Tag(\mathbb{K}, \mathbb{M})$ *is the mac generation algorithm which returns a tag* $\tau = \{x_1, x_2, ..., x_s\}$ *where each* x_i *is the solution of the following linear system in* $GF(2^b)$*:*

$$\begin{cases} KD_1(k_1, m_1) \cdot x_1 + KD_2(k_1, m_1) \cdot x_2 + ... + KD_s(k_1, m_1) \cdot x_s \equiv MAC_{k_1}(m_1) \\ KD_1(k_2, m_2) \cdot x_1 + KD_2(k_2, m_2) \cdot x_2 + ... + KD_s(k_2, m_2) \cdot x_s \equiv MAC_{k_2}(m_2) \\ ... \\ KD_1(k_s, m_s) \cdot x_1 + KD_2(k_s, m_s) \cdot x_2 + ... + KD_s(k_s, m_s) \cdot x_s \equiv MAC_{k_s}(m_s) \end{cases}$$

Here b is polynomial in the security parameter ℓ *and KD stands for a key derivation process. If such a solution does not exist, then the M-MAC algorithm fails and returns* $\perp$*.*

3. $v \leftarrow Ver(k, m, \tau)$ *is the verification algorithm which returns* 1 *if and only if having* $\tau' = MAC_k(m)$ *it holds* $\tau' \equiv KD_1(k, m) \cdot x_1 + KD_2(k, m) \cdot x_2 + ... + KD_s(k, m) \cdot x_s$*. Otherwise it returns* 0*.*

Let us emphasize that the probability that the *M-MAC* fails to return a solution is negligible in the security parameter (if proper b and s are chosen). As shown in [22] the probability that an n by n matrix with random elements from $GF(q)$ is non-singular converges to $\prod_{i=1}^{\infty}(1 - 1/q^i)$ as $n \to \infty$. For example in case when $s = 4$ we have a chance of around 10^{-5} for $b = 16$ and 10^{-10} for $b = 32$ for the *M-MAC* to fail.

Example 2. We want to clarify here our intentions on *M-MAC*s with respect to the protocol design. Consider a case when master M broadcasts messages m_1 and m_2 to slaves S_1, S_2 along with the authentication tag. To increase efficiency of our protocol we want to authenticate both messages with the same mixed MAC and more, since only a portion of each tag is disclosed (reducing the bus overhead but also the security level), we want one of the slaves to be able to carry out the authentication further with a new part of a valid tag (note that this is what happens in the case of the two-stage authentication). Consider that the following packets arrive on the bus: message m_1, message m_2 and the mixed tag obtained by simply concatenating the two tags $MAC_{k_1}(m_1)||MAC_{k_2}(m_2)$. However, due to the message filtering of the CAN bus it may be that the two messages do not reach both slaves. Assume message m_1 reaches S_1 and m_2 reaches S_2. Now neither S_1 or S_2 can carry the authentication further, even in the case when they both have k_1 and k_2 they are not in possession of the message that reached the other slave and thus they can not validate the other part of the tag. More relevant, note that the nodes are unable to detect if the other part of the tag is compromised. Now consider the case of the *LM-MAC*. In this case the tag is obtained by mixing the two tags via the linear equation system, e.g., the two

components of the tag x_1, x_2 verify a relation of the form $\alpha_1 x_1 + \alpha_2 x_2 = MAC_{k_1}(m_1)$ and $\beta_1 x_1 + \beta_2 x_2 = MAC_{k_2}(m_2)$ (here α's and β's are derived from the secret keys k_1, k_2). If an adversary compromises any part of the tag, i.e., either x_1 or x_2, then both equations will fail to verify and any of the receivers can detect this (indeed, we assume that the adversary is not in possession of the secret keys k_1 and k_2 since in such case he can compute correct *LM-MAC*s anyway). Consequently, with the *LM-MAC*s any of them can check the tag for correctness and this validation will also hold for the other receiver, this is inherited from the strong non-malleability property for *M-MAC*s.

We now sketch a more formal account of the properties that we require for our building blocks. These are mediated by two attack games against unforgeability, i.e., $Game^{UF}_{M\text{-}MAC}$, and strong non-malleability, i.e., $Game^{SNM}_{M\text{-}MAC}$. Both games are defined for a generic *M-MAC* construction and in particular the *LM-MAC* can be proved to resist such attacks. The attack game on strong non-malleability $Game^{SNM}_{M\text{-}MAC}$ against an *M-MAC* requires an adversary to be able to construct an *M-MAC* in such way that verification fails with at least one of the keys but succeeds with another. An *M-MAC* that is resilient to such an attack is called *strongly non-malleable*.

Definition 1. (Unforgeability Attack Game) We define the *M-MAC* unforgeability game $Game^{UF}_{M\text{-}MAC}$ as the following five stage game between challenger $\mathcal{C}$ and adversary Adv:

1.Challenger $\mathcal{C}$ runs the key generation algorithm $Gen(1^\ell, s)$ to get a key set $\mathbb{K} = \{k_0, k_1, ..., k_s\}$.
2.Adversary Adv is allowed to requests $\mathcal{C}$ any subset of the keyset $\mathbb{K}' = \{k_{j_0}, k_{j_1}, ..., k_{j_t}\}$, $t < s$ where $\forall j_i \in [1..s]$. That is, the adversary is always missing at least 1 of the keys.
3.Adversary Adv is allowed to make queries to the MAC generation oracle $\mathcal{O}^{Tag}(\mathbb{K}, \mathbb{M})$ for any message tuple $\mathbb{M}$ to obtain the corresponding tag $\tau \leftarrow Tag(\mathbb{K}, \mathbb{M})$ and to the verification oracle $\mathcal{O}^{Ver}(i, \tau, m)$ with any key index i, tag τ and message m and the oracle will return 1 if and only if τ is a correct tag under key k_i for message m.
4.Eventually, the adversary outputs the tuple $(m^\lozenge, \tau^\lozenge, i)$ for some index i such that he is not in possession of k_i.
5.The game output is 1 if the following two conditions hold: Ver outputs 1 on (τ, m, k_i) and the adversary never queried m to the Tag oracle. Otherwise the game output is 0.

Definition 2. (Unforgeability) We say that a mixed message authentication code *M-MAC* is unforgeable if: $\Pr\left[Game^{UF}_{M\text{-}MAC}(1^\ell, s) = 1\right] < \text{negl}(\ell)$.

Definition 3. (Strong Non-malleability Attack Game) We define the *M-MAC* strong non-malleability game $Game^{SNM}_{M\text{-}MAC}$ as the following five stage game between challenger $\mathcal{C}$ and adversary Adv:

1.Challenger $\mathcal{C}$ runs the key generation algorithm $Gen(1^\ell, s)$ to get a key set $\mathbb{K} = \{k_0, k_1, ..., k_s\}$.
2.Adversary Adv is allowed to requests $\mathcal{C}$ any subset of the keyset $\mathbb{K}' = \{k_{j_0}, k_{j_1}, ..., k_{j_t}\}$, $t < s - 1$ where $\forall j_i \in [1..s]$. That is, the adversary is always missing at least 2 of the

keys.
3.Adversary *Adv* is allowed to make queries to the MAC generation oracle $\mathcal{O}^{Tag}(\mathbb{K}, \mathbb{M})$ for any message tuple $\mathbb{M}$ to obtain the corresponding tag $\tau \leftarrow Tag(\mathbb{K}, \mathbb{M})$ and to the verification oracle $\mathcal{O}^{Ver}(i, \tau, m)$ with any key index i, tag τ and message m and the oracle will return 1 if and only if τ is correct tag under key k_i for message m.
4.Eventually, the adversary outputs the pair $(m^{\Diamond}, \tau^{\Diamond})$.
5.The game output is 1 if there are at least two keys $k, k' \in \mathbb{K}$ such that the following two conditions hold: *Ver* outputs 1 on (τ, m, k) but outputs 0 on (τ, m, k') and the keys k, k' are not part of the adversary keyset $\mathbb{K}'$. Otherwise the game output is 0.

Definition 4. (Strong Non-malleability) We say that a mixed message authentication code *M-MAC* is strongly non-malleable if: $\Pr\left[Game^{SNM}_{M\text{-}MAC}(1^{\ell}, s) = 1\right] < \text{negl}(\ell)$.

Due to space limitations, a proof of this theorem in the random oracle model is deferred for the extended version of this work.

4.4.2 Experimental results

To evaluate the performance of the proposed protocol suite, we used several setups with different hardware components to determine the minimum authentication delay. Automotive grade embedded devices from Freescale and Infineon as well as a notebook equipped with an adapter for CAN communication from Vector were employed to build the nodes of our experimental CAN network. The embedded platforms that we used are representatives for industry's low-end and high-end edges.

4.4.2.1 Test beds

Using the aforementioned components we built several test beds. First, the case of a system using the centralized authentication approach with one master node and 4 slave nodes was considered:

- *Testbed 1: S12X+4×S12X*. Both master and slave nodes are built on identical S12X development boards with CAN communication speed set to 125kbps.
- *Testbed 2: TC1782+4×TC1797*. Master and slave nodes are built on similar TriCore development boards having the same computational and communication capabilities. CAN communication speed is set to 1Mbps.
- *Testbed 3: Intel T7700+4×S12X*. The master node is implemented on a PC (Intel Core2Duo CPU T7700@2.4GHz) while slave nodes are built on the S12X boards. The master-slave CAN communication is done through the CANcardXL using a low speed CANcab for 125kbps.
- *Testbed 4: Intel T7700+4×TC1797*. The master node is implemented on the same PC as in the previous case while slave nodes are built on the TriCore platform. This time a high speed CANcab is used with the CANcardXL to enable a 1Mbps communication speed.

A different testbed was set up to compare the various proposed variants of the key splitting protocol on a system with 8 slaves based on S12X nodes. Two variants were considered as we further discus: centralized authentication (in this case one extra node was added to act as the master) and two-stage authentication.

4.4.2.2 Protocol performance

Centralized authentication was implemented on the four testbeds prepared for this purpose. Our implementation considers 6 groups of two nodes each formed by combining the four available nodes. Messages and authentication tags are always sent as separate frames and the message size is always 8 bits. The MAC size for each group is set to 21 bits so that 3 authentication tags fit a single 64 bit CAN frame. The MAC is computed using the MD5 hash function over an input formed by concatenating the group key to the message. The resulting hash is then truncated to the desired size. Table 4.10 holds the timings and bus loads for each test bed. Here δ is the authentication delay, i.e., the time needed by a node to authenticate the message once it receives it. For the bus load we considered the fraction of traffic caused by the authentication tags over the entire bandwidth.

Master	Slave	δ	Bitrate	Bus load
S12X	4xS12X	2.54 *ms*	125 kbps	53.84%
PC	4xS12X	1.848 *ms*	125 kbps	72.22%
TriCore	4xTriCore	267 μs	1 Mbps	54.31%
PC	4xTriCore	378 μs	1 Mbps	42.54%

Table 4.10: Centralized authentication with 4 nodes

As expected, scenarios in which high end devices played the role of master nodes (PC, TriCore) showed better performance than in the case of low end master nodes. The case of a PC master with TriCore slaves does not perform better, despite the considerable difference in computational power between master nodes (TriCore vs. Intel Core2Duo) due to limitations of CAN adapters. Because of their internal hardware/software design, these adapters introduce some limitations, e.g., the average response time specified by Vector for the CANcardXL is 100μs.

To evaluate the protocol behavior when using different trade-offs we implemented different variants of the key splitting authentication protocol on a system with 8 slaves built on S12X nodes. By grouping the eight nodes two by two we obtain a total of 28 groups. The size of the authentication tags and the truncated MAC size differ in each variant. We set up the implementations as follows:

- *Centralized:* The message sending node computes and sends one MAC for each group that he is part of. The master computes and sends one MAC for each of the other 21 groups (if groups of size 2 are used). If the master is to perform the authentication in only 2 frames then each MAC can be truncated to 5 bits and this will lead to a total of 35 security bits for each node. But if we increase the number of authentication frames

from the master to 3, then each MAC can be truncated to 9 bits giving a total of 63 authentication bits for each node which is a reasonable level for real-time security.

• *Two-stage:* The master node is missing in this implementation, therefore we use two helper nodes for computing and sending the complete authentication tag. In the two-stage variant, the sender node will first put one authentication tag on the bus which contains the full 36 authentication bits for one of the helper nodes, 20 bits for the second one and 8 extra bits for another node. This first tag is followed by a second tag generated by the first helper node which contains the remaining 16 authentication bits for the second helper node and 48 bits equally distributed for three of the remaining nodes. To complete the 36 authentication bits for each of the remaining nodes, the sender node and the second helper node will each put an authentication tag on the bus. As discussed previously, the security level can be raised to around 64 bits by using groups of size 3 and the described tag allocation procedure.

Table 4.11 holds the results achieved with these two implementations. The worst performer in terms of authentication delay is the implementation of the centralized authentication variant as it involves computing MACs for each of the 28 groups in a sequential manner. In the other implementation, a smaller number of MACs are computed some of which are done by different nodes in parallel. A smaller authentication delay is obtained when using the two-stage implementation at the cost of an increased CPU load on the sender side. However, this cost is somewhat compensated by the higher level of security offered by the fact that the sender node offers more authentication bits.

Variant	**Master**	**Slave**	δ	**Bus load**
Centralized	S12X	8xS12X	22.624 *ms*	11.27%
Two-stage	-	8xS12X	6.806 *ms*	46.21%

Table 4.11: Centralized & Cascade with 8 nodes

4.4.2.3 Computational performance with linearly mixed tags

The results from Tables 4.10 and 4.11 use the simple concatenation of individual MACs computed with MD5 as the underlying hash function. We now take a brief account of the impact of mixing tags using linear systems of equations, complete experimental results on this will be available in the extended version of our work, here we make an accurate estimation of the computational costs. To begin with, in Table 4.12 we give an overview on the computational timings for various hash functions and input sizes on both of the employed platforms. For the Linearly Mixed MACs, in addition to the computation of the MACs, two supplemental computational tasks are required: solving the linear system of equations on the sender side (a task which should be usually done by the master which has higher computational power) and reconstructing the MAC on the receiver side. Our experimental results obtained on the communication master equipped with the Intel 2.4GHz core with the well known NTL

Hash function	**Input size (bytes)**					
	S12			TriCore		
	0	16	64	0	16	64
MD5	371μs	374μs	1414μs	10.16μs	11.00μs	18.34μs
SHA-1	1.144ms	1.148ms	4.510ms	14.64μs	15.10μs	27.60μs
SHA-256	2.755ms	2.755ms	5.440ms	41.70μs	42.35μs	80.80μs

Table 4.12: Computational performance of employed embedded platforms

library (http://www.shoup.net/ntl/) showed that the computational cost of solving the system for 2 nodes in $GF(2^8)$ up to $GF(2^{32})$ are around 3–6 times more intensive than an MD5 computation and this increases to 10–20 times the MD5 computation in the case of 4 nodes. Since this task should be done by the master node it shouldn't raise computational issues. The reconstruction of the MAC was around 10 times cheaper compared to the linear mixing procedure and compared to MD5 it was in the range of 0.5–5 times more intensive, the later in the case of 8 nodes and $GF(2^{32})$. All these are reasonable amounts of computations and we believe that they can be significantly improved with platform dependent tweaks.

5 Physical layer authentication

Alternatives for assuring source identification on CAN are limited. To alleviate this, here we take an entirely distinct approach by trying to identify CAN nodes based on their signal patterns and present results contained in [87]. Personal contributions consists of identifying detection mechanisms and evaluating their capabilities on two chosen CAN transceiver models. This opens road to the possibility of building an intrusion detection system (IDS), an instrument that has been successfully used in computer networks.

Most IDSs built for computer networks focus on the data layer and need databases of considerable size to fulfill their purpose. These kind of IDSs would not be suitable for automotive networks given the usual memory and computational constraints of microcontrollers used in automotive applications. We therefore have to find other means of detecting the presence of intruders. The answer may be in the unique physical characteristics of electronic circuits that have been frequently used to identify individual devices. By fingerprinting all network nodes that send messages on the bus, the authenticity of sent messages could be assured and rogue nodes could be spotted since their pattern would not match any of the known fingerprints.

Establishing the identity of a node based on signal characteristics has obviously both pros and cons. The great advantage behind this technique, is in the fact that it doesn't require any modification of the protocol that is already running. Clearly, any crypto-based solution requires modifications at the protocol layer which triggers backward compatibility issues. Computational and communication constraints would be also encountered (as shown in previous chapters), a main cause for the current absence of cryptography from in-vehicle networks. The alternative that we investigate here would simply require the presence of an additional node that is able to analyze signal patterns and trigger an alarm whenever something goes wrong. This alarm can either be used immediately, or it can serve as a clue for forensic purposes. Using this method to aid forensics may prove to be very useful for building an efficient event data recorder (EDR), a device that resembles the airplane black box which will become mandatory in all new U.S. cars starting from 2015. Another advantage is that, while the aim of our work here is to analyze whether the technique is feasible or not on CAN networks, the same technique may be applied to other bus types inside vehicles, e.g., LIN, FlexRay, etc. Thus the approach is general and can be easily applied to other communication buses. On the down side, the technique is vulnerable to changes in the circuitry, e.g., power fluctuations, overloads, etc. Here we made our experiments in a laboratory setup that is clearly less exposed to the environment than a real-world automotive network. But as the results that we obtained are positive, i.e., they show

that we could clearly distinguish between certain CAN nodes, this gives us motivation to pursue as future work an analysis for the behaviors of nodes inside a real car (such experiments are currently out of reach for us).

This chapter is organized as follows. In section 5.1 we start with presenting related work before proceeding in section 5.2 with the description of the methods employed for source identification. Section 5.3 holds experimental results for using the proposed methods.

5.1 Related work

The subject of intrusion detection was intensively studied in areas like computer networks, wireless sensor networks [63] or vehicular ad-hoc networks (VANETs) [129]. However, until recently, the subject was neglected in relation to in-vehicle networks.

When considering the type of analysis done on collected data, existing IDSs can be classified as signature-based or anomaly-based. A signature-based IDS [2] looks for sequences of events or data patterns in the analyzed data. Their efficiency depends on the constant updating of their rule database, so they will not be able to detect novel attacks. An anomaly-based IDS [39] attempts to detect anomalies in the behavior of the system based on an estimation on what should be the normal behavior.

Depending on the source of information used for detection, IDSs can be classified as host or network-based. Host-based IDSs work by analyzing system events (e.g., system calls) on the monitored device, while network-based IDSs look at network related events (e.g., network traffic). Most network-based IDSs collect their data starting from the data link layer up to higher layers.

Solutions, similar to the ones used in computer network IDSs, were applied for CAN bus traffic. These approaches involve analyzing the network traffic in search for abnormal behavior (e.g. additional or incorrect traffic) [90], [88], [17]. Adequate processing power and memory space have to be provided for these methods to be effective, making their usability in constraint environments questionable on the short term.

Hard to control manufacturing inconsistencies generate minute differences in the signals generated by identical electronic circuits and this aspect can be used for device identification which in turn stands as a basis for intrusion detection. Hall et al. [51] used the transient portion of radio signals to build transceiver fingerprints, while Hotelling's T^2 and a threshold were used to determine if the transceiverprint matches the profile of any known transceiver. Similar techniques were also applied in other areas where wireless communication is employed such as sensor networks [15] and bluetooth communications [52]. Beamforming and artificial noise were used in wireless networks in a physical layer approach to provide secure communication in the presence of eavesdroppers [109]. Device identification based on physical signal characteristics was investigated in the case of wired buses as well. The work done by Gerdes [41] focused on identifying Ethernet cards by studying the synchronization signal (found at the beginning of each Ethernet frame) with the help of a matched

filter. Reported experimental results show that Ethernet cards of different models can be easily distinguished while for cads of the same model an acceptable degree of accuracy can be achieved with adequate preprocessing of input data. This is strong evidence that physical characteristics can be exploited on in-vehicle buses as well. However, to the best of our knowledge, dedicated automotive buses, such as CAN, were not studied in this regard.

5.2 Methodology

The physical representation of an actual CAN messages consists in the signal that is generated by a dedicated transceiver. These transceivers are produced by various manufacturers according to the CAN specifications [64]. This specification allows great freedom in the implementation of the physical layer signaling behavior in order to allow application dependent optimizations. As a consequence, signals produced by transceivers from different manufacturers are not identical. Moreover, as each electronic component gathers unique physical characteristics (even when produced by the same manufacturer), the signals generated by CAN transceivers show up a unique behavior.

5.2.1 Data acquisition

For fingerprinting CAN transceivers based on their signaling behavior a common portion of the CAN frame has to be chosen as a basis for comparison. The CAN ID that can be found in the arbitration field of each CAN frame could be suitable for this purpose. This field is used to reflect the type of data contained in the message or to denote the sender node. Hence, for a given application, the CAN IDs that can be sent by a particular node are known in advance. Any rogue device that wants to impersonate a honest node will have to send a message with an already allocated ID. Therefore, we chose the portion of the CAN frame representing the message identifier to be stored as fingerprint data. By correlating signal fingerprints with CAN IDs an IDS could detect if the sender of the CAN frame is indeed authorized to transmit such messages.

Figure 5.1 illustrates a set of four oscilloscope plots showing the first bit of several CAN frames. This bit represents the start of frame which is marked by a rise in the voltage level on the bus and is followed by the message identifier. The captured frames have the message identifier set to 000h and were sent at a baudrate of 10KBit/s. The plot covers a period of 100 μs during which 1000 samples were taken. Two frames (Frame 1 and 2) are sent by the same transceiver and a third one (Frame 3) is sent by a different transceiver but of the same type (a PCA82C251 transceiver from NXP). The fourth frame (Frame 4) is sent by a third transceiver of a different model (a TJA1054T also from NXP). As expected, CAN signals generated by distinctly built transceivers are easy to separate (Frames 1–3 vs. Frame 4) but this is not the case for frames from identical transceivers (Frames 1–3) that are very similar and distinguishing between

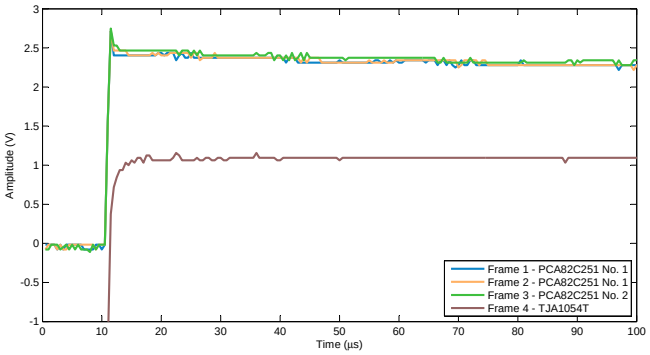


Figure 5.1: Section of arbitration fields from CAN frames produced by three transceivers

them is not straightforward (note that these signals greatly overlap). On a basic visual inspection, differences between signals produced by the same transceiver have the same magnitude with the ones by different transceivers from the same make and model.

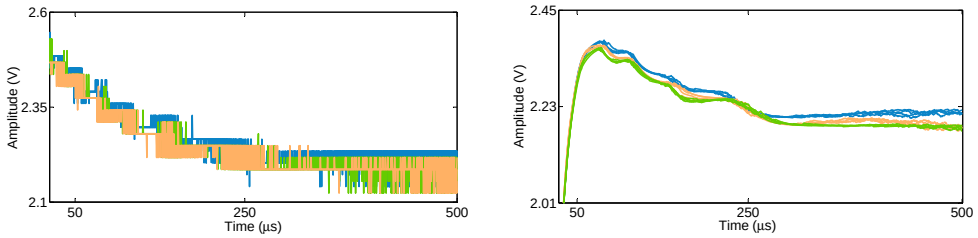
5.2.2 Signal processing tools

We briefly describe the mathematical tools that we used in order to obtain a finer grain analysis of transceiver's characteristics. In what follows we will use the following notations: the *fingerprint* $\mathcal{F}$ is the reference data stored for each device, the *signature* *Sig* represents a fresh data pending for verification, ℓ is the length of the previous two vectors. Indexes of $\mathcal{F}$ and *Sig* denote a particular dataset on which they are computed.

5.2.2.1 Low-pass filtering

Signatures extracted from CAN frames have to be compared with stored fingerprints. As shown in Figure 5.2(a) the signal is noisy and a simple bit-by-bit comparison between two signals may not be even possible as signals greatly overlap. As a first step, we filter the acquired signal in order to remove, as much as possible, from the unwanted noise. Figure 5.2 illustrates signals representing the start of the arbitration fields (frames have the same characteristics as the ones in Figure 5.1) from 3 transceivers with very similar behavior both in an unfiltered and filtered form. The overlapping is abundant with the unfiltered signals. The filtering that we use is a low-pass filter as described by the following equation: $\widetilde{Sig_\alpha}[i] = \lambda \cdot Sig_\alpha[i] + (1-\lambda) \cdot \widetilde{Sig_\alpha}[i-1], i = 1..\ell$, ($\lambda$ is the smoothing factor). This basic filtering technique produces a noticeable difference between the signals, but it is not yet enough to distinguishing between very similar ones.

While we used a software implementation, the low-pass filtering also has the advantage that it can be implemented in hardware, thus reducing the computational



(a) (b)

Figure 5.2: Arbitration fields from three transceivers before low-pass filtering (left) and after low-pass filtering (right)

costs. The following techniques will be also applied on the unfiltered signal (in this case the tilde notation is omitted).

5.2.2.2 Mean squared error

The mean squared error (MSE) is a method of quantifying the difference between two sets of values.

For applying MSE in our case we consider the stored fingerprint as the reference set and the signal to be checked as the second set. The mean squared error is computed as: $MSE(Sig_\alpha, \mathcal{F}_\beta) = \frac{1}{\ell}\sum_{i=1}^{\ell}\left(Sig_\alpha[i] - \mathcal{F}_\beta[i]\right)^2$. MSE can either be computed over the signal to be verified for all the stored fingerprints corresponding to the specified CAN ID, or else, the signature can be compared with the average of the stored fingerprints. If the signal being verified comes from one of the authorized nodes then the MSE computed with the node's fingerprint should have the lowest value while all other MSE computations should result in greater values.

5.2.2.3 Convolution

One common issue in signal processing is signal misalignment. Even when using a costly oscilloscope (as we did for our tests) one cannot guarantee that the sampled signals are perfectly aligned. This problem can be alleviated by convolving the two compared signals. The convolution operation is equivalent with the multiplication of two polynomials. In our case, the signal sample vector holds the polynomial coefficients. The result of this operation is a vector of length $n+m-1$, where m and n are the lengths of the two input vectors. The input vectors have the same length ℓ in our case thus the length of the result is $2\ell-1$. Each element of the resulted vector (CONV) is defined as: $CONV_k(Sig_\alpha, \mathcal{F}_\beta) = \sum_{i=1}^{\ell}(\mathcal{F}_\beta[i] \cdot Sig_\alpha[k-i+1])$. We consider the maximum value of the convolution vector as the measure of the signal similitude to the reference because this represents the point of best alignment between the two signals. Therefore, the value used for comparison is: $MAXCONV = \max_{k=1}^{2\ell-1}(\sum_{i=1}^{\ell}(\mathcal{F}_\beta[i] \cdot Sig_\alpha[k-i+1]))$.

To identify sender nodes, a large set of training data should be gathered to define variation intervals as fingerprints for each known transceiver. Upon receiving a new

frame, its signature will be checked to see if it fits one of the stored intervals.

5.3 Experimental results

We performed our tests on CAN frames produced by transceivers mounted on two different types of devices: USB-to-CAN adapters and embedded development boards. These are described next along with the results.

5.3.1 Experimental setup

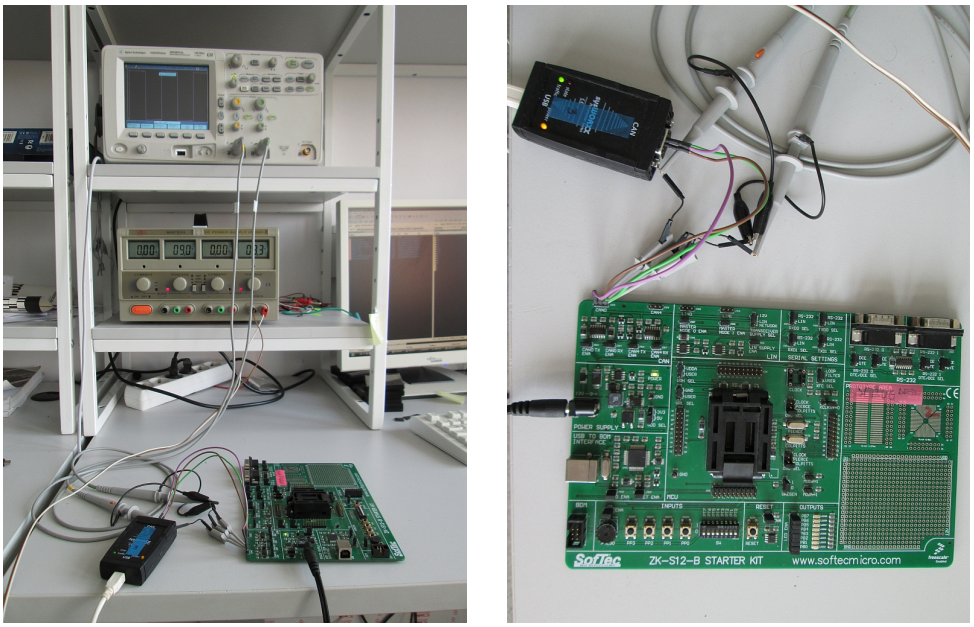
The SYS TEC USB-CANmodul1, a single channel USB to CAN interfacing device equipped with a PCA82C251 transceiver, was used as the USB-to-CAN adapter. The PCA82C251 is a CAN transceiver built for 24 volt systems and capable of high speed communication (up to 1MBaud). To minimize the influence of computer components on the CAN signal (due to power fluctuations) we assured a fixed 5V external power supply for the USB-CANmoduls.

The ZK-S12-B development board was the second type of device used in our experiments. It is built around a Freescale automotive type microcontroller from the S12 family and fitted with two TJA1054T fault tolerant low speed (up to 125kBaud) transceivers for CAN communication.

Sets of frames produced by each individual device were collected using an oscilloscope based automatic acquisition setup. The acquisition of the frames was done using an Agilent MSO6012A oscilloscope with a sample rate of 2 GSa/s and a resolution of up to 12 bits. The oscilloscope probes were connected to the CAN-H and CAN-L bus lines as depicted in Figure 5.3. The sampled differential CAN signals were saved using Matlab and Agilent device drivers on a PC connected to the oscilloscope. Each captured signal is composed of a number of 1000 sample points as this was the maximum achievable in the normal acquisition mode. The sample sets were acquired over a period of several hours (approximately 4 hours and 40 minutes for 100k signals) in which the target device was set to continuously send the same message ID on the bus. Between 20000 and 100000 signals were saved during each acquisition session. Multiple acquisition sessions were made for the same devices to test the stability of the signal characteristics over time.

5.3.2 Source identification

In order to test the source identification techniques, we collected thousands of CAN frames from a series of 10 USB-to-CAN devices and 5 S12 development boards (each board has 2 transceivers and each of them was sampled). Before collecting the signals, each device was labeled for convenient identification. Differences between the signals produced by different types of transceivers are clearly visible even without applying additional processing techniques. For transceivers of the same type, signal similarities are considerable and require additional processing of the acquired data.



(a) (b)

Figure 5.3: Experimental Setup: oscilloscope acquiring CAN frames from the bus (a) and close-up with the USB-CANmodul and S12 board connection (b)

The main set of signals was generated having the CAN ID set to 0x000 and at a speed of 10Kbaud.

In some cases, we tried to filter the noise in order to enhance specific signal particularities. Unfortunately, instead of being easier to differentiate, filtered signals produced by different transceivers were at times more similar as the filtering dimmed specific signal characteristics along with the noise. To facilitate a visual analysis we built plots holding the resulted signatures of all transceivers of a certain type. In what follows we show a set of such plots which mark the worst case scenario, i.e., the reference signal is very similar to signals produced by several other transceivers. The collected signal data was processed in Matlab for computing the signal signature using both the MSE and convolution approaches as discussed next.

5.3.2.1 MSE based separation

Figure 5.4 presents MSE values computed for a series of PCA82C251 transceivers (from the USB-to-CAN modules), numbered from 1 to 10, using as a reference the fingerprint for $\mathbf{T}^4_{\text{USB}}$ (which is the fourth module). For each of the transceivers 20000 frames were captured. The band containing MSE values for signals produced by $\mathbf{T}^4_{\text{USB}}$ (red) is situated in the lower part of the plot suggesting good similarities. The majority of transceivers, with two exceptions, produce less similar signals. One exception comes

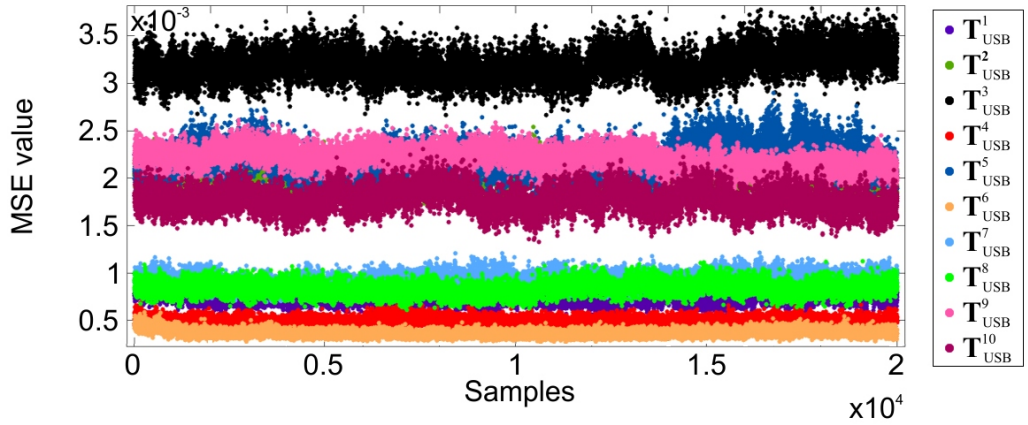


Figure 5.4: 20000 MSE values computed for each PCA82C251 transceiver having the signature of $\mathbf{T}^4_{USB}$ as the reference

from $\mathbf{T}^1_{USB}$ (purple) which generates signatures that overlap with values from the band for $\mathbf{T}^4_{USB}$. The signature band for $\mathbf{T}^6_{USB}$ (orange) is situated lower on the graph and could be falsely deemed as the better fitting for the fingerprint of $\mathbf{T}^4_{USB}$. In the case of TJA1054T transceivers (Figure 5.5) there are three transceivers that produce signals very similar to the reference signal (produced by $\mathbf{T}^{2''}_{S12}$ which is the second transceiver from the second development board) as seen in the zoomed window. The difference between the signals from the other four transceivers and the reference is in this case greater than for the USB-to-CAN modules.

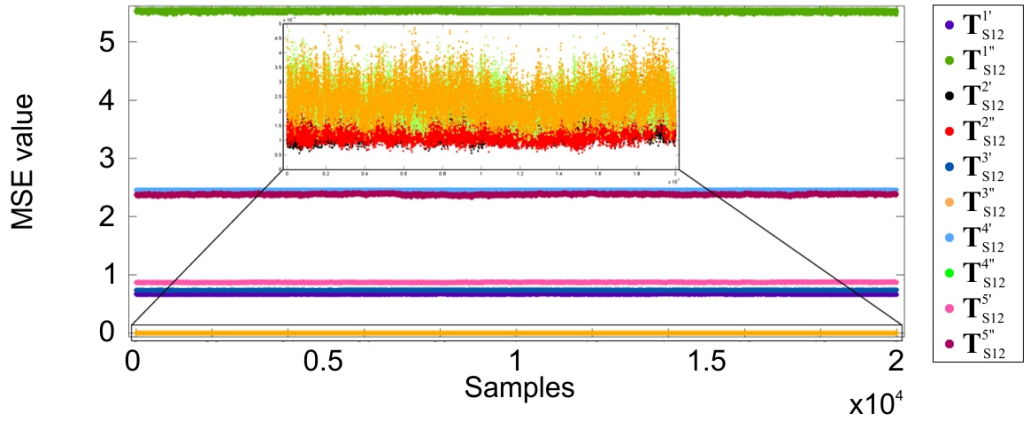


Figure 5.5: 20000 MSE values computed for each TJA1054T transceiver having the signature of $\mathbf{T}^{2''}_{S12}$ as the reference

5.3.2.2 Convolution based separation

Figure 5.6 shows values obtained using convolutions over the same data set as in the previous section. The band containing values for signals produced by $\mathbf{T}^4_{USB}$ is constantly overlapped by the one generated for $\mathbf{T}^6_{USB}$ while the band for $\mathbf{T}^7_{USB}$ is just occasionally overlapping it. The rest of the transceivers produce results that can be clearly distinguished from the target $\mathbf{T}^4_{USB}$. When applying the convolution on the signals from the S12 board, results are similar as in the MSE case. Three transceivers are hard to distinguish from the true target based on the signals they generate, while the other six transceivers clearly generate different signals as can be seen in Figure 5.7.

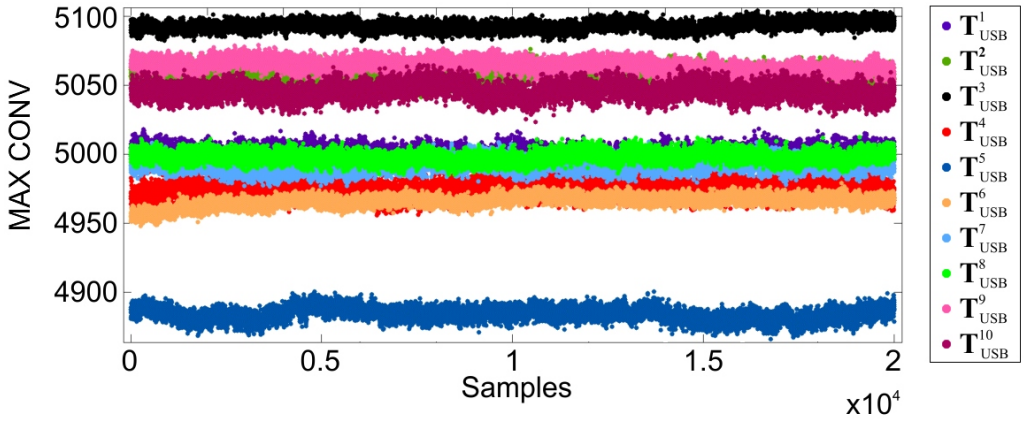


Figure 5.6: 20000 convolved values computed for each PCA82C251 transceiver having the signature of $\mathbf{T}^4_{USB}$ as the reference

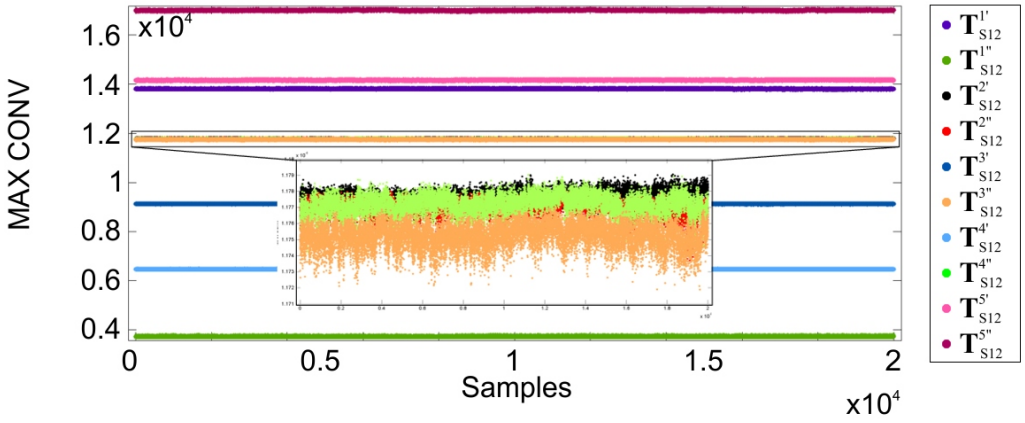


Figure 5.7: 20000 convolved values computed for each TJA1054T transceiver having the signature of $\mathbf{T}^{2''}_{S12}$ as the reference

5.3.2.3 Mean-value based separation

As reflected by the plots presented above, in some cases the signature of a single signal might not be enough to accurately determine its source. To increase detection accuracy we tested identification based on multiple signal signatures by computing a signature mean value and comparing it with a fingerprint obtained in the same manner. We superimposed a line representing the mean value of collected signatures over each set presented in the previous plots. Figures 5.8 through 5.11 contain signal signatures of the reference signals and signatures very similar to them along with the mean values for each data set. In the case of USB-to-CAN modules, data mean values

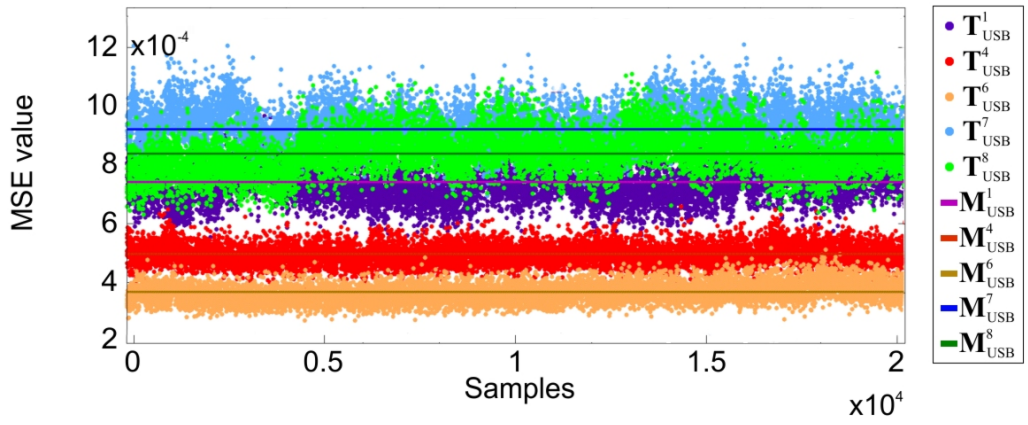


Figure 5.8: Mean of MSE values computed for PCA82C251 transceivers with signatures similar to the $\mathbf{T}^{4}_{USB}$ reference fingerprint

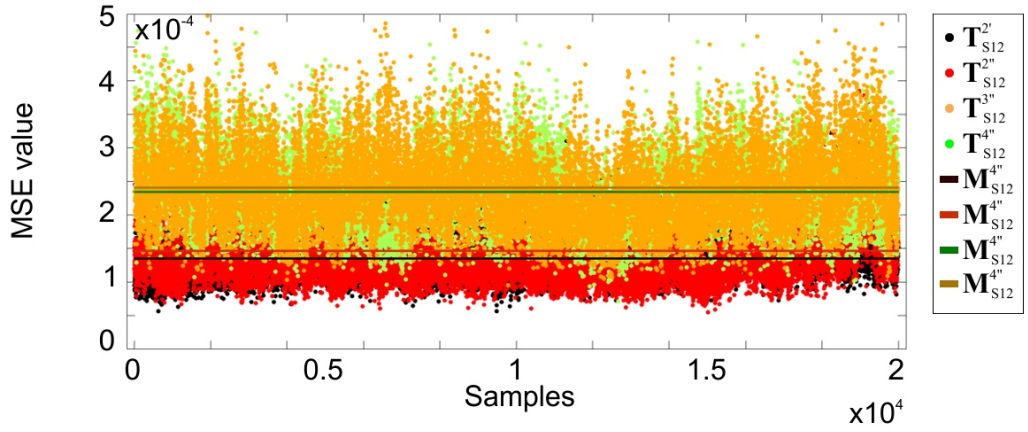


Figure 5.9: Mean MSE values computed for TJA1054T transceivers with signatures similar to the $\mathbf{T}^{2''}_{S12}$ reference fingerprint

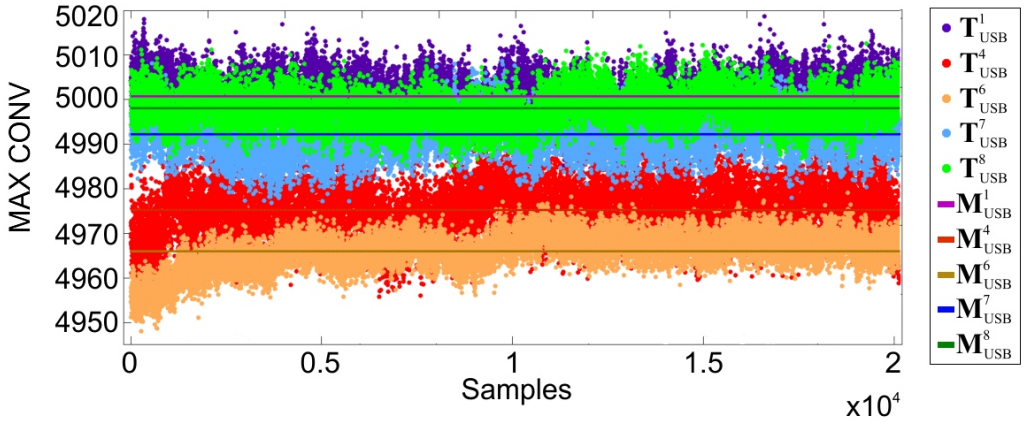


Figure 5.10: Mean of convolved values computed for PCA82C251 transceivers with signatures similar to the $\mathbf{T}^{4}_{USB}$ reference fingerprint

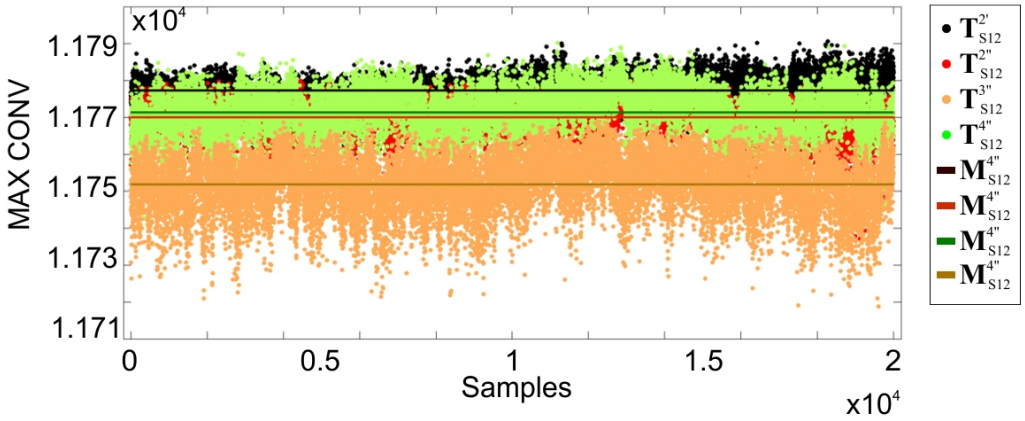


Figure 5.11: Mean of convolved values computed for TJA1054T transceivers with signatures similar to the $\mathbf{T}^{2''}_{S12}$ reference fingerprint

makes it easier to distinguish between overlapping signatures. However, the plots holding TJA1054T data show signature mean values to be much closer to each other, making source identification difficult. The actual identification accuracy depends on the number of signatures used to compute the mean value and the signal stability over time.

5.3.3 Identification success rate

As the identification success rates for using MSE are very similar to the ones obtained when using convolution based separation we include only results for the MSE

case.

Tables 5.1 and 5.2 illustrate the identification rates obtained in our tests when using MSE based detection. Each cell contains the identification result for a transceiver against a target transceiver designated as the first entry in each row. Cell values range from 0 (transceiver was not identified as being the target transceiver) to 1 (tested transceiver is the target) and were computed from 20000 signals for each transceiver. For easy reading, we use the mark ✓ whenever there was no confusion between the two transceivers. Thresholds were empirically defined for each target transceiver based on the MSE value ranges so that the detection rate of each target transceiver is not smaller than 90%. We allowed rates smaller than 100% for transceivers with similar behavior when the bands of MSE values overlapped.

Target	$\mathbf{T}^{1}_{USB}$	$\mathbf{T}^{2}_{USB}$	$\mathbf{T}^{3}_{USB}$	$\mathbf{T}^{4}_{USB}$	$\mathbf{T}^{5}_{USB}$	$\mathbf{T}^{6}_{USB}$	$\mathbf{T}^{7}_{USB}$	$\mathbf{T}^{8}_{USB}$	$\mathbf{T}^{9}_{USB}$	$\mathbf{T}^{10}_{USB}$
$\mathbf{T}^{1}_{USB}$	0.995	✓	✓	0.001	✓	✓	✓	✓	✓	✓
$\mathbf{T}^{2}_{USB}$	✓	0.920	✓	✓	✓	✓	✓	✓	0.695	0.003
$\mathbf{T}^{3}_{USB}$	✓	✓	1	✓	✓	✓	✓	✓	✓	✓
$\mathbf{T}^{4}_{USB}$	0.001	✓	✓	0.985	✓	0.151	✓	✓	✓	✓
$\mathbf{T}^{5}_{USB}$	✓	✓	✓	✓	1	✓	✓	0.002	✓	✓
$\mathbf{T}^{6}_{USB}$	✓	✓	✓	0.001	✓	0.999	✓	✓	✓	✓
$\mathbf{T}^{7}_{USB}$	✓	✓	✓	✓	✓	✓	0.941	0.004	✓	✓
$\mathbf{T}^{8}_{USB}$	✓	✓	✓	✓	✓	✓	0.002	0.967	✓	✓
$\mathbf{T}^{9}_{USB}$	✓	0.437	0.001	✓	✓	✓	✓	✓	0.994	✓
$\mathbf{T}^{10}_{USB}$	✓	0.047	✓	✓	✓	✓	✓	✓	✓	0.997

Table 5.1: Identification rates for PCA82C251 (ID set to 0x000) when using MSE

Target	$\mathbf{T}^{1'}_{S12}$	$\mathbf{T}^{1''}_{S12}$	$\mathbf{T}^{2'}_{S12}$	$\mathbf{T}^{2''}_{S12}$	$\mathbf{T}^{3'}_{S12}$	$\mathbf{T}^{3''}_{S12}$	$\mathbf{T}^{4'}_{S12}$	$\mathbf{T}^{4''}_{S12}$	$\mathbf{T}^{5'}_{S12}$	$\mathbf{T}^{5''}_{S12}$
$\mathbf{T}^{1'}_{S12}$	1	✓	✓	✓	✓	✓	✓	✓	✓	✓
$\mathbf{T}^{1''}_{S12}$	✓	1	✓	✓	✓	✓	✓	✓	✓	✓
$\mathbf{T}^{2'}_{S12}$	✓	✓	0.908	0.876	✓	0.204	✓	0.029	✓	✓
$\mathbf{T}^{2''}_{S12}$	✓	✓	0.831	0.901	✓	0.118	✓	0.121	✓	✓
$\mathbf{T}^{3'}_{S12}$	✓	✓	✓	✓	1	✓	✓	✓	✓	✓
$\mathbf{T}^{3''}_{S12}$	✓	✓	0.999	0.991	✓	0.901	✓	0.300	✓	✓
$\mathbf{T}^{4'}_{S12}$	✓	✓	✓	✓	✓	✓	1	✓	✓	✓
$\mathbf{T}^{4''}_{S12}$	✓	✓	0.527	0.934	✓	0.029	✓	0.900	✓	✓
$\mathbf{T}^{5'}_{S12}$	✓	✓	✓	✓	✓	✓	✓	✓	1	✓
$\mathbf{T}^{5''}_{S12}$	✓	✓	✓	✓	✓	✓	✓	✓	✓	1

Table 5.2: Identification rates for TJA1054T (ID set to 0x000) when using MSE

The results shown in these tables come as a support for the previously depicted graphical representations. The batch of PCA82C251 transceivers seem to be easier to distinguish with a maximum false detection rate of 69.5% ($\mathbf{T}^{2}_{USB}$ vs. $\mathbf{T}^{9}_{USB}$) while in the case of some of our TJA1054T transceiver this range can go up to 99.9% ($\mathbf{T}^{2'}_{S12}$ vs. $\mathbf{T}^{3''}_{S12}$) which means that similarities in signaling behavior are too big to make a correct identification. However, this confusion can be apparently circumvented by

clever allocation of the IDs as we discuss in the next section.

Using mean-value based separation increases the identification accuracy but only at the cost of more signal acquisitions which makes it less appealing for practical scenarios. For example, in the case of TJA1054T, computing the mean on only 100 signals (Table 5.3) did not bring major improvements, these were more obvious for mean values computed on 1000 frames (Table 5.4) or more. However, due to great similarities it is still hard to distinguish between some transceivers such as the case of $\mathbf{T}^{2'}_{S12}$ and $\mathbf{T}^{2''}_{S12}$.

Target	$\mathbf{T}^{1'}_{S12}$	$\mathbf{T}^{1''}_{S12}$	$\mathbf{T}^{2'}_{S12}$	$\mathbf{T}^{2''}_{S12}$	$\mathbf{T}^{3'}_{S12}$	$\mathbf{T}^{3''}_{S12}$	$\mathbf{T}^{4'}_{S12}$	$\mathbf{T}^{4''}_{S12}$	$\mathbf{T}^{5'}_{S12}$	$\mathbf{T}^{5''}_{S12}$
$\mathbf{T}^{1'}_{S12}$	1	✓	✓	✓	✓	✓	✓	✓	✓	✓
$\mathbf{T}^{1''}_{S12}$	✓	1	✓	✓	✓	✓	✓	✓	✓	✓
$\mathbf{T}^{2'}_{S12}$	✓	✓	0.900	0.959	✓	0.308	✓	0.007	✓	✓
$\mathbf{T}^{2''}_{S12}$	✓	✓	0.567	0.900	✓	0.003	✓	0.008	✓	✓
$\mathbf{T}^{3'}_{S12}$	✓	✓	✓	✓	1	✓	✓	✓	✓	✓
$\mathbf{T}^{3''}_{S12}$	✓	✓	0.811	0.973	✓	0.900	✓	0.222	✓	✓
$\mathbf{T}^{4'}_{S12}$	✓	✓	✓	✓	✓	✓	1	✓	✓	✓
$\mathbf{T}^{4''}_{S12}$	✓	✓	0.277	0.968	✓	✓	✓	0.900	✓	✓
$\mathbf{T}^{5'}_{S12}$	✓	✓	✓	✓	✓	✓	✓	✓	1	✓
$\mathbf{T}^{5''}_{S12}$	✓	✓	✓	✓	✓	✓	✓	✓	✓	1

Table 5.3: Identification rates for TJA1054T (ID set to 0x000) when using MSE median of 100 signals

Target	$\mathbf{T}^{1'}_{S12}$	$\mathbf{T}^{1''}_{S12}$	$\mathbf{T}^{2'}_{S12}$	$\mathbf{T}^{2''}_{S12}$	$\mathbf{T}^{3'}_{S12}$	$\mathbf{T}^{3''}_{S12}$	$\mathbf{T}^{4'}_{S12}$	$\mathbf{T}^{4''}_{S12}$	$\mathbf{T}^{5'}_{S12}$	$\mathbf{T}^{5''}_{S12}$
$\mathbf{T}^{1'}_{S12}$	1	✓	✓	✓	✓	✓	✓	✓	✓	✓
$\mathbf{T}^{1''}_{S12}$	✓	1	✓	✓	✓	✓	✓	✓	✓	✓
$\mathbf{T}^{2'}_{S12}$	✓	✓	0.900	0.927	✓	0.160	✓	✓	✓	✓
$\mathbf{T}^{2''}_{S12}$	✓	✓	0.037	0.905	✓	✓	✓	✓	✓	✓
$\mathbf{T}^{3'}_{S12}$	✓	✓	✓	✓	1	✓	✓	✓	✓	✓
$\mathbf{T}^{3''}_{S12}$	✓	✓	0.300	0.512	✓	0.900	✓	0.160	✓	✓
$\mathbf{T}^{4'}_{S12}$	✓	✓	✓	✓	✓	✓	1	✓	✓	✓
$\mathbf{T}^{4''}_{S12}$	✓	✓	0.130	0.927	✓	✓	✓	0.900	✓	✓
$\mathbf{T}^{5'}_{S12}$	✓	✓	✓	✓	✓	✓	✓	✓	1	✓
$\mathbf{T}^{5''}_{S12}$	✓	✓	✓	✓	✓	✓	✓	✓	✓	1

Table 5.4: Identification rates for TJA1054T (ID set to 0x000) when using MSE median of 1000 signals

5.3.4 Behavior for various IDs and baud rates

To determine if the message ID has any influence on the source identification process we also acquired sets of signals with different CAN IDs. By applying the above mentioned methods on these sample sets, most of the results were similar to the case of CAN ID 0x000. However, both advantageous and disadvantageous

modifications can appear as the set of nodes whose signatures overlaps is subject to change. For example, in the case when we set the ID to 0x555, we still had three transceivers that have signatures similar to the one of $\mathbf{T}_{S12}^{2''}$ but not the same ones as when using ID 0x000. This can be exploited in a constructive manner: the signature similarities between a target and other transceivers can be minimized by selecting an appropriate set of IDs. Figure 5.12 illustrates MSE values for signals produced by TJA1054T transceivers with the CAN ID set to 0x555.

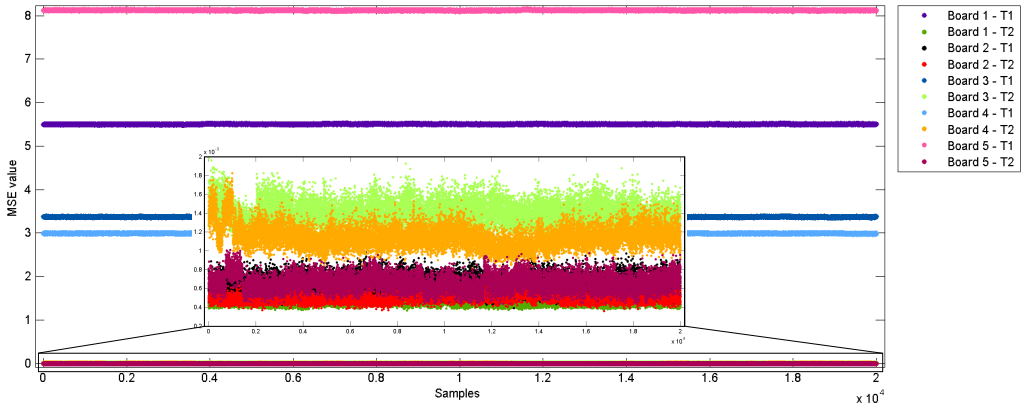


Figure 5.12: 20000 MSE values computed for each TJA1054T transceiver having the signature of $\mathbf{T}_{S12}^{2''}$ as the reference and the ID set to 0x555

Communication speed can also influence signaling behavior. In Figure 5.13 the MSE values were computed for the same CAN messages as in our previous example (i.e., with the ID set to 0x555) but at a different speed, i.e., 125kbaud. In this case

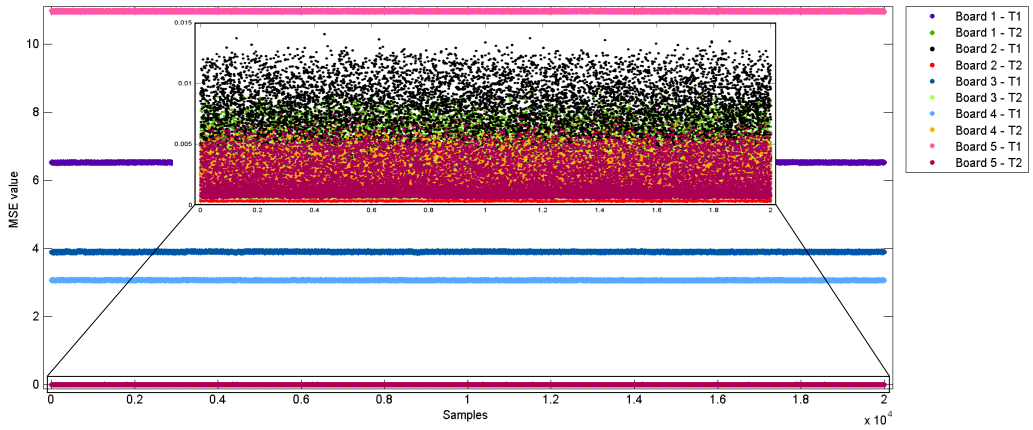


Figure 5.13: 20000 MSE values computed for TJA1054T with the signature of $\mathbf{T}_{S12}^{2''}$ as the reference, the ID set to 0x555 and the baudrate configured to 125kbaud

the picture drawn by transceiver signatures is similar with the one obtained when using lower communication speeds suggesting that the ID alone is responsible for the change in results. On the downside, the similar signatures are now even harder to distinguish indicating that, at higher speeds, detection capabilities are decreased but this can be compensated by employing faster ADCs to achieve smaller sampling periods.

As a consequence of the influence of changing message IDs on transceiver signaling behavior the identification rate is also affected. Table 5.5 shows the identification rate obtained for TJA1054T transceivers when using the ID to 0x555 and the baud-rate to 125k. At a glance, identification rate seems to worsen, however, on a closer inspection it turns out that for transceivers $\mathbf{T}^{2'}_{S12}$ and $\mathbf{T}^{3''}_{S12}$, that were hardly distinguishable, the confusion rate now drops to 0%. This strongly suggests that clever allocation of IDs for specific transceivers can yield extremely high identification rates.

Target	$\mathbf{T}^{1'}_{S12}$	$\mathbf{T}^{1''}_{S12}$	$\mathbf{T}^{2'}_{S12}$	$\mathbf{T}^{2''}_{S12}$	$\mathbf{T}^{3'}_{S12}$	$\mathbf{T}^{3''}_{S12}$	$\mathbf{T}^{4'}_{S12}$	$\mathbf{T}^{4''}_{S12}$	$\mathbf{T}^{5'}_{S12}$	$\mathbf{T}^{5''}_{S12}$
$\mathbf{T}^{1'}_{S12}$	1	✓	✓	✓	✓	✓	✓	✓	✓	✓
$\mathbf{T}^{1''}_{S12}$	✓	0.916	✓	0.004	✓	0.369	✓	✓	✓	✓
$\mathbf{T}^{2'}_{S12}$	✓	✓	0.950	0.250	✓	✓	✓	0.452	✓	✓
$\mathbf{T}^{2''}_{S12}$	✓	0.078	0.263	0.950	✓	0.777	✓	0.767	✓	0.722
$\mathbf{T}^{3'}_{S12}$	✓	✓	✓	✓	1	✓	✓	✓	✓	✓
$\mathbf{T}^{3''}_{S12}$	✓	0.426	✓	0.784	✓	0.950	✓	0.527	✓	0.817
$\mathbf{T}^{4'}_{S12}$	✓	✓	✓	✓	✓	✓	1	✓	✓	✓
$\mathbf{T}^{4''}_{S12}$	✓	✓	0.545	0.817	✓	0.580	✓	0.950	✓	0.510
$\mathbf{T}^{5'}_{S12}$	✓	✓	✓	✓	✓	✓	✓	✓	1	✓
$\mathbf{T}^{5''}_{S12}$	✓	0.002	0.003	0.750	✓	0.879	✓	0.484	✓	0.950

Table 5.5: Identification rates for TJA1054T (ID set to 0x555)

Since collisions between transceiver patterns appear to be random, they should be precisely linked to the resolution of the measurements (in our case, a 12-bit resolution). Consequently, this will bound the success probability of an adversary that inserts its own device on the network. If the available resolution is too low, one should consider increasing the size of the ID or even extending the identification to other portions of the frames.

5.3.5 Signal drift in time

The signaling behavior of electronic circuits may exhibit slight changes over time. To address potential issue, we repeated our experiments over a period of several months and found the signatures to remain within certain boundaries. In Figure 5.14 we draw plots containing the multiple signature bands of the same transceiver that were acquired on different dates (roughly separated by weeks or even months) along with bands of transceivers with very similar signaling behavior. Our tests included continuous signal acquisition from devices over a period of roughly one month of uninterrupted functioning. No noticeable drift was visible during these tests. Indeed,

over longer periods of time, it is possible for drifts to appear but we consider that this effect can be compensated by continuously updating the fingerprint of each device by using newly received signals.

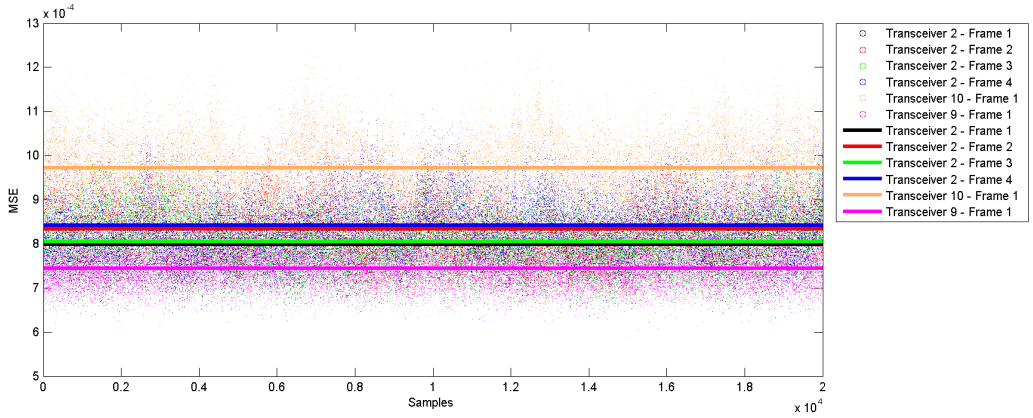


Figure 5.14: Signature variation in time

Additional drifts in signal characteristics can be induced by other electronic circuits that are part of the system equipped with CAN transceivers. However, these influences, provided that they are constant, can even benefit the identification process for the case of similar transceivers that are placed on distinct circuits. Even if all vehicle subsystems are equipped with transceivers from the same manufacturer, these subsystems should be different enough to bring unique influences to the CAN signal produced. On the downside, if the neighbouring electronic circuits only cause temporary disturbances in the signaling behavior of the device this can have negative influences on the identification capabilities.

6 Conclusions

In-vehicle security is an emerging topic in the broader context of information security, its importance stems from the numerous reported attacks. Protection must be assured for various assets such as human safety, intellectual property, car components or even the car itself. In some cases security mechanisms are completely absent while in other even mechanisms that already exist are vulnerable to attacks. Usually, the security mechanisms inside cars are aimed at preventing unauthorized access to in-vehicle buses. Once penetrated in-vehicle buses offer little to no protection against attacks. This threat can be alleviated by introducing authentication protocols to in-vehicle networks such as the widely used CAN.

The performance analysis in Chapter 3 illustrates the constraint nature of some automotive-specific microcontrollers by evaluating their performance in relation to hash functions. The execution speed is bounded by the CPU architecture and reduced frequencies that characterize these platforms. Memory is also an issue as it is usually too limited to hold both the ECU firmware and a memory consuming cryptographic protocol. The performance can be enhanced when using devices equipped with multiple cores or cryptographic co-processors. Still, the overall performance of automotive microcontrollers is still too low to enable the deployment of mechanisms commonly used in computer networks without affecting system timing constraints. System constraints can be compensated by devising lightweight cryptographic primitives.

As the CAN protocol provides no support for authenticating communication our first contribution exhibits mechanisms at the application layer. We found the well known TESLA protocol to be a suitable candidate and different trade-offs and schemes were studied in order to determine an optimal choice of parameters. By the nature of this family of authentication protocols, authentication delays are introduced. In the best case we were able to obtain authentication with delays in the order of several hundreds micro-seconds (these are not very realistic due to various limitations of underlying electronics, e.g. oscillator tolerance). Obviously the performance greatly depends on the microcontrollers being employed, e.g. the TriCore microcontroller clearly outperformed the S12X and the authentication delays dropped by almost two orders of magnitude. This approach may not be suitable for some applications but we do expect that the delays achieved here are suitable in many practical real-time scenarios.

One-time signatures are the alternative for assuring immediate authentication in CAN networks. This is more prominent in the case of messages with reduced size. The main purpose of our work was to assess which are the limitations of such a solution in CAN networks and our experimental results showed up to several kilobits of authen-

ticated traffic can be achieved while the delay at which messages are authenticated can be lower to tens of milliseconds.

LiBrA, the third approach that we employed is based on splitting keys between groups of nodes. The proposed protocol is efficient when the number of nodes is low and the corrupted nodes are in minority. We expect this to be the case in many automotive scenarios where, although the number of ECUs may be high, the numbers of manufacturers from which they come may not be high and distributing trust between several groups is an acceptable solution. If the number of nodes is too high we see only two resolutions: public key cryptography (with the drawback of high computational requirements, at least 2 orders of magnitude) or TESLA like protocols with the drawback of authentication delays.

When the application layer approach is not a viable alternative, the physical layer may be used to provide source authentication. The methodology we described here proved to be usable in distinguishing the source of messages without any modifications on the software that the node is running or of the network that it is part of. This may set a distinct perspective on assuring broadcast authentication in CAN networks, an environment where cryptographic techniques are currently absent and will be hard to implement due to the various constraints. Besides detecting intrusions, this technique may also be useful for forensic purposes as event data recorders are closer to be mandatory for newly produced automobiles.

While the approaches presented here were tested on CAN buses they could also be used on other buses such as FlexRay or BroadR-Reach (emerging Ethernet physical layer standard designed for use in automotive applications).

The results presented here were published in a series of papers co-authored by the author of this thesis. The author's contributions are summarized in what follows:

1. **Comprehensive survey of adversary models, attack surfaces and reported attacks on in-vehicle networks.** To establish a solid background and justification for the thesis proposal we investigate existing literature on in-vehicle security issues and present a compilation of reported attacks in section 2.2.

2. **Performance analysis of automotive-specific microcontrollers for security applications.** The performance achievable when using automotive embedded devices to implement hash functions is illustrated by Chapter 3 which holds the results published in [85] and [86].

3. **Performance improvements trough parallelism and hardware acceleration.** Implementation details on how parallelism and cryptographic co-processors can be used to enhance the performance of a series of hash functions are also presented in Chapter 3 and [85, 86].

4. **Proof of concept implementation of several TESLA-based protocols to obtain concrete results on a CAN setup.** A first evaluation of broadcast authentication using TESLA-based protocols on CAN buses is given. The minimum

achievable authentication delay should fit the constraints of most in-vehicle systems but it may be too high for some safety critical systems such as the airbag. Section 4.2 is dedicated to this topic and holds results published in [44], [46], [47].

5. **Implementation and performance evaluation of LiBrA-CAN protocols.** A new protocol based on key splitting between groups of nodes is implemented in several variants and their performance is presented. The authentication delays are improved compared to the case of using TESLA. Section 4.4 holds details of these results which were published in [49].

6. **Development of a method for identifying sender nodes in CAN networks by analyzing physical signal characteristics.** Analyzing signal characteristics proves to be an efficient method when comparing signals from different types of transceivers and can also offer acceptable results in case CAN nodes are implemented using transceivers of different makes and models. The thesis covers this subject in Chapter 5 and the results are published in [87].

A decision on what protocol should be used in real-world vehicular applications can be taken only by manufacturers and by means of consortium. The results presented here open road in this direction by proposing new protocols or trade-offs and giving clear hints on the constraints and performances that can be achieved.

A Appendix A - Results obtained during the PhD studies

A.1 Papers published in ISI indexed publications

- Groza, B., Murvay, P.-S., *Broadcast Authentication in a Low Speed Controller Area Network*, E-Business and Telecommunications, pp. 330-344, Springer Berlin Heidelberg, 2012.
- Groza, B., Minea, M., Cristea, M., Murvay, P.-S., Iacob, M., *Protocol vulnerabilities in practice: Causes, modeling and automatic detection*, Proceedings of the Romanian Academy Series A - Mathematics, Physics, Technical sciences, Information science, Vol. 13, Issue 2, pp. 167-174, 2012. (Impact Factor - 0.537, Relative Influence Score - 0.191)
- Groza, B., Murvay, P.-S., *Efficient Protocols for Secure Broadcast in Controller Area Networks*, IEEE Transactions on Industrial Informatics, Vol. 9, Issue 4, pp. 2034-2042, 2013. (Impact Factor - 3.381, Relative Influence Score - 1.607)
- Murvay, P.-S., Groza, B., *Source Identification Using Signal Characteristics in Controller Area Networks*, Accepted for publication in IEEE Signal Processing Letters, 2014. (Impact Factor - 1.674, Relative Influence Score - 1.910)

A.2 Papers published in other indexed publications

- Groza, B., Murvay, P.-S., *Higher Layer Authentication for Broadcast in Controller Area Networks (CAN)*, The Sixth International Conference on Security and Cryptography (SECRYPT'11), 2011.
- Groza, B., Murvay, P.-S., *Secure Broadcast With One-Time Signatures In Controller Area Networks*, The Sixth International Conference on Availability, Reliability and Security (ARES'11), 2011.
- Murvay, P.-S., Groza, B., *Performance improvements for SHA-3 finalists by exploiting microcontroller on-chip parallelism*, The Sixth International Conference on Risks and Security of Internet and Systems (CRiSIS'11), 2011.
- Groza, B., Murvay, P.-S., van Herrewege, A., Verbauwhede, I., *LiBrA-CAN: a Lightweight Broadcast Authentication protocol for Controller Area Networks*, The 11th International Conference on Cryptology and Network Security (CANS'12), 2012.

- Groza, B., Murvay., P.-S., *Secure Broadcast with One-Time Signatures in Controller Area Networks*, International Journal of Mobile Computing and Multimedia Communications, Vol. 5, Issue 3, pp. 1-18, 2013.

- Murvay. P.-S., Groza, B., *Performance Evaluation of SHA-2 Standard vs. SHA-3 Finalists on two Freescale Platforms*, International Journal of Secure Software Engineering, Vol. 4, Issue 4, 2013, pending publication.

A.3 Papers published in other non-indexed volumes

- Murvay, P.-S., *Cryptography on embedded devices with application to in-vehicle communication*, Advanced Computer Architecture and Compilation for High-Performance and Embedded Systems summer school (ACACES'11), pp. 309-312, 2011.

A.4 Other activities

- Attended the Advanced Computer Architecture and Compilation for High-Performance and Embedded Systems (ACACES) 2011 summer school, at Fiuggy, Italy, in july 2011

- Reviewer for the 7th International Conference on Risks and Security of Internet and Systems (CRiSIS), 2012.

- Invited reviewer for Journal of Loss Prevention in the Process Industries, Elsevier, 2012.

References

[1] R. Anderson, F. Bergadano, B. Crispo, J.-H. Lee, C. Manifavas, and R. Needham. A new family of authentication protocols. *SIGOPS Oper. Syst. Rev.*, 32:9–20, October 1998.

[2] F. Anjum, D. Subhadrabandhu, and S. Sarkar. Signature based intrusion detection for wireless ad-hoc networks: A comparative study of various routing protocols. In *Vehicular Technology Conference, 2003. VTC 2003-Fall. 2003 IEEE 58th*, volume 3, pages 2152–2156. IEEE, 2003.

[3] A. Antunes, F. Dias, and A. Mota. Can-based real time adaptive distributed control. In *Proc. 8th International CAN Conference*, 2002.

[4] K. Aoki, G. Roland, Y. Sasaki, and M. Schläffer. Byte Slicing Grøstl - Optimized Intel AES-NI and 8-bit Implementations of the SHA-3 Finalist Grøstl. In *Proceedings of The Sixth International Conference on Security and Cryptography, SECRYPT 2011*. SciTePress, 2011.

[5] H. K. Aslan. A hybrid scheme for multicast authentication over lossy networks. *Computers & Security*, 23(8):705 – 713, 2004.

[6] AUDI AG. *Audi A8'10 Bordnetz und Vernetzung*.

[7] J.-P. Aumasson, L. Henzen, W. Meier, and R. C.-W. Phan. SHA-3 proposal BLAKE. Submission to NIST (Round 3), 2010.

[8] J.-P. Aumasson, S. Neves, Z. Wilcox-O'Hearn, and C. Winnerlein. Blake2: simpler, smaller, fast as md5. accessed 29 December 2012, [Online]. Available: https://blake2.net/blake2_20121223.pdf, 2012.

[9] H. Bar-El. Intra-vehicle information security framework. In *Proceedings of 9th Embedded Security in Cars Conference, ESCAR*, 2009.

[10] A. Barisani and B. Daniele. Unusual car navigation tricks: Injecting rds-tmc traffic information signals. In *Proceedings of the CanSecWest Conference*, 2007.

[11] M. Bellare, R. Canetti, and H. Krawczyk. Keying hash functions for message authentication. In *Advances in Cryptologyâ CRYPTOâ 96*, pages 1–15. Springer, 1996.

[12] D. Berbecaru, L. Albertalli, and A. Lioy. The forwarddiffsig scheme for multicast authentication. *IEEE/ACM Trans. Netw.*, 18:1855–1868, December 2010.

[13] F. Bergadano, D. Cavagnino, and B. Crispo. Individual authentication in multiparty communications. *Computers & Security*, 21(8):719 – 735, 2002.

[14] G. Bertoni, J. Daemen, M. Peeters, and G. V. Assche. The Keccak SHA-3 submission. Submission to NIST (Round 3), 2011.

[15] K. Bonne Rasmussen and S. Capkun. Implications of radio fingerprinting on the security of sensor networks. In *Security and Privacy in Communications Networks and the Workshops, 2007. SecureComm 2007. Third International Conference on*, pages 331–340. IEEE, 2007.

[16] G. Bourdon, P. Ruaux, and S. Delaplace. Can for autonomous mobile robot. In *Proceedings of the 3rd International CAN Conference*, 1996.

[17] M. Brooks. Anomaly detection on vehicle networks. In *escar, Embedded Security in Cars USA Conference*, 2013.

[18] R. Bruckmeier. Ethernet for automotive applications. In *Freescale Technology Forum, Orlando*, June 2010.

[19] R. Buchty, N. Heintze, and D. Oliva. Cryptonite a programmable crypto processor architecture for high-bandwidth applications. In C. Moeller-Schloer, T. Ungerer, and B. Bauer, editors, *Organic and Pervasive Computing ARCS 2004*, volume 2981 of *Lecture Notes in Computer Science*, pages 184–198. Springer Berlin / Heidelberg, 2004.

[20] G. Cena, L. Durante, and A. Valenzano. A new can-like field network based on a star topology. *Computer Standards & Interfaces*, 23(3):209–222, 2001.

[21] Y. Challal, A. Bouabdallah, and H. Bettahar. H2a: Hybrid hash-chaining scheme for adaptive multicast source authentication of media-streaming. *Computers & Security*, 24(1):57 – 68, 2005.

[22] L. S. Charlap, H. D. Rees, and D. P. Robbins. The asymptotic probability that a random biased matrix is invertible. *Discrete Mathematics*, 82(2):153 – 163, 1990.

[23] 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. *Proceedings of the 2011 Usenix Security*, 2011.

[24] M. Cheminod, A. Pironti, and R. Sisto. Formal vulnerability analysis of a security system for remote fieldbus access. *Industrial Informatics, IEEE Transactions on*, 7(1):30–40, 2011.

[25] D. Coppersmith and M. Jakobsson. Almost optimal hash sequence traversal. In *Proceedings of the 6th international conference on Financial cryptography*, FC'02, pages 102–119, Berlin, Heidelberg, 2003. Springer-Verlag.

[26] S. David. Subsea instrumentation interface standardization in the offshore oil and gas industry. In *Proc. 11th International CAN Conference*, 2006.

[27] D. Dolev and A. Yao. On the security of public key protocols. *Information Theory, IEEE Transactions on*, 29(2):198–208, 1983.

[28] ECRYPT. The SHA-3 Zoo. http://ehash.iaik.tugraz.at/wiki/The_SHA-3_Zoo, 2011.

[29] ECRYPT. Measurements of SHA-3 finalists, indexed by machine. accessed 29 December 2012, [Online]. Available: http://bench.cr.yp.to/results-sha3.html, 2012.

[30] K. Etschberger, I. Automation, D.-I. C. Schlegel, and O. Schnelle. Canopen maritime–a new standard for highly dependable communication systems. In *9th international CAN Conference, iCC*, 2003.

[31] K. Etschberger, R. Hofmann, A. Neuner, U. Weissenrieder, and B. Wiulsroed. A failure-tolerant canopen system for marine automation systems. In *7th International CAN Conference, CiA, vol. Safety*, 2000.

[32] N. Falliere, L. O. Murchu, and E. Chien. W32.Stuxnet dossier. Technical report, Symantec, 2011.

[33] G. Farrell, A. Tseloni, and N. Tilley. The effectiveness of vehicle security devices and their role in the crime drop. *Criminology and Criminal Justice*, 11(1):21–35, 2011.

[34] N. Ferguson, S. Lucks, B. Schneier, D. Whiting, M. Bellare, T. Kohno, J. Callas, and J. Walker. The Skein Hash Function Family. Submission to NIST (Round 3), 2010.

[35] M. Fischlin. Fast verification of hash chains. In *RSA Security Cryptographer's Track 2004*, volume 2964 of *Lecture Notes in Computer Science*, pages 339–352. Springer-Verlag, 2004.

[36] A. Francillon, B. Danev, and S. Capkun. Relay attacks on passive keyless entry and start systems in modern cars. In *Proceedings of NDSS*, 2010.

[37] Freescale. *MC9S12XDP512 Data Sheet, Rev. 2.21, October 2009, [Online]. Available: http://www.freescale.com/files/microcontrollers/doc/data_sheet/MC9S12XDP512RMV2.pdf*, 2009.

[38] Freescale. *K60 Sub-Family Reference Manual, Rev. 5, 21 May 2011, [Online]. Available: http://http://www.freescale.com/files /microcontrollers/doc/ref_manual/K60P144M150SF3RM.pdf*, 2011.

[39] P. Garcia-Teodoro, J. Diaz-Verdejo, G. Maciá-Fernández, and E. Vázquez. Anomaly-based network intrusion detection: Techniques, systems and challenges. *computers & security*, 28(1):18–28, 2009.

[40] P. Gauravaram, L. R. Knudsen, K. Matusiewicz, F. Mendel, C. Rechberger, M. SchlÃ ffer, and S. S. Thomsen. Grøstl – a SHA-3 candidate. Submission to NIST (Round 3), 2011.

[41] R. M. Gerdes, M. Mina, S. F. Russell, and T. E. Daniels. Physical-layer identification of wired ethernet devices. *IEEE Transactions on Information Forensics and Security*, 7(4):1339–1353, 2012.

[42] J. Großschädl and E. Sava . Instruction set extensions for fast arithmetic in finite fields GF (p) and GF (2 m). In *Cryptographic Hardware and Embedded Systrems - CHES 2004*, pages 133–147. Springer Verlag, LNCS 3156, 2004.

[43] B. Groza, M. Minea, M. Cristea, P.-S. Murvay, and M. Iacob. Protocol vulnerabilities in practice: causes, modeling and automatic detection. *Proceedings of the Romanian Academy, Series A*, 13(2):167–174, 2012.

[44] B. Groza and P.-S. Murvay. Higher layer authentication for broadcast in controller area networks. In *SECRYPT*, pages 188–197, 2011.

[45] B. Groza and P.-S. Murvay. Secure broadcast with one-time signatures in controller area networks. In *Availability, Reliability and Security (ARES), 2011 Sixth International Conference on*, pages 371–376. IEEE, 2011.

[46] B. Groza and P.-S. Murvay. Broadcast authentication in a low speed controller area network. In *E-Business and Telecommunications*, pages 330–344. Springer, 2012.

[47] B. Groza and P.-S. Murvay. Efficient protocols for secure broadcast in controller area networks. *Industrial Informatics, IEEE Transactions on*, 9(4):2034–2042, 2013.

[48] B. Groza and P.-S. Murvay. Secure broadcast with one-time signatures in controller area networks. *International Journal of Mobile Computing and Multimedia Communications (IJMCMC)*, 5(2):1–8, 2013.

[49] B. Groza, P.-S. Murvay, A. Van Herrewege, and I. Verbauwhede. Libra-can: a lightweight broadcast authentication protocol for controller area networks. In *Cryptology and Network Security*, pages 185–200. Springer, 2012.

[50] R. A. Gupta and M.-Y. Chow. Networked control system: Overview and research trends. *Industrial Electronics, IEEE Transactions on*, 57(7):2527–2535, 2010.

[51] J. Hall, M. Barbeau, and E. Kranakis. Radio frequency fingerprinting for intrusion detection in wireless networks. *IEEE Transactions on Defendable and Secure Computing*, 2005.

[52] J. Hall, M. Barbeau, and E. Kranakis. Detecting rogue devices in bluetooth networks using radio frequency fingerprinting. In *In IASTED International Conference on Communications and Computer Networks*. Citeseer, 2006.

[53] F. Hartwich. CAN with Flexible Data-Rate. In *14th international CAN conference*, 2012.

[54] C. Hoder, T. Sumers, and G. Zulauf. Hot-Wiring of the Future: Exploring Automotive CANs. In *REcon Conference*, 2013.

[55] T. Hoppe and J. Dittman. Sniffing/replay attacks on can buses: A simulated attack on the electric window lift classified using an adapted cert taxonomy. In *Proceedings of the 2nd Workshop on Embedded Systems Security (WESS)*, 2007.

[56] T. Hoppe, S. Kiltz, and J. Dittmann. Security threats to automotive can networks–practical examples and selected short-term countermeasures. *Computer Safety, Reliability, and Security*, pages 235–248, 2008.

[57] T. Hoppe, S. Kiltz, and J. Dittmann. Automotive it-security as a challenge: Basic attacks from the black box perspective on the example of privacy threats. In *Computer Safety, Reliability, and Security*, pages 145–158. Springer, 2009.

[58] Y.-C. Hu, M. Jakobsson, and A. Perrig. Efficient constructions for one-way hash chains. In J. Ioannidis, A. Keromytis, and M. Yung, editors, *Applied Cryptography and Network Security*, volume 3531 of *Lecture Notes in Computer Science*, pages 167–190. Springer Berlin / Heidelberg, 2005.

[59] E. II. Yearly report on algorithms and keysizes (2009-2010), revision 1.0, 2011.

[60] S. Indesteege, N. Keller, O. Dunkelman, E. Biham, and B. Preneel. A practical attack on keeloq. In *Advances in Cryptology–EUROCRYPT 2008*, pages 1–18. Springer, 2008.

[61] Infineon. *TC1784 32-Bit Single-Chip Microcontroller, User's Manual, V1.0, 2009-07*, 2009.

[62] Infineon. *TC1797 32-Bit Single-Chip Microcontroller, User's Manual, V1.1, 2009-05*, 2009.

[63] K. Ioannis, T. Dimitriou, and F. C. Freiling. Towards intrusion detection in wireless sensor networks. In *Proc. of the 13th European Wireless Conference*. Citeseer, 2007.

[64] Road Vehicles - Controller Area Network (CAN), 2003.

[65] M. Jakobsson. Fractal hash sequence representation and traversal. In *Proceedings of the 2002 IEEE International Symposium on Information Theory (ISIT '02)*, pages 437–444, 2002.

[66] K. Kaneko, K. Harada, F. Kanehiro, G. Miyamori, and K. Akachi. Humanoid robot hrp-3. In *Intelligent Robots and Systems, 2008. IROS 2008. IEEE/RSJ International Conference on*, pages 2471–2478. IEEE, 2008.

[67] C. Karlof, N. Sastry, and D. Wagner. Tinysec: a link layer security architecture for wireless sensor networks. In *Proceedings of the 2nd international conference on Embedded networked sensor systems*, SenSys '04, pages 162–175, New York, NY, USA, 2004. ACM.

[68] S. Kino. Application and benefits of an open devicenet control system in the forest products industry. In *Pulp and Paper, 1999. Industry Technical Conference Record of 1999 Annual*, pages 196–203. IEEE, 1999.

[69] P. Kleberger, T. Olovsson, and E. Jonsson. Security aspects of the in-vehicle network in the connected car. In *Intelligent Vehicles Symposium (IV), 2011 IEEE*, pages 528–533. IEEE, 2011.

[70] K. Koscher, A. Czeskis, F. Roesner, S. Patel, T. Kohno, S. Checkoway, D. McCoy, B. Kantor, D. Anderson, H. Shacham, et al. Experimental security analysis of a modern automobile. In *Security and Privacy (SP), 2010 IEEE Symposium on*, pages 447–462. IEEE, 2010.

[71] L. Lamport. Password authentication with insecure communication. *Commun. ACM*, 24:770–772, November 1981.

[72] W. Lawrenz. *CAN system engineering: from theory to practical applications*, volume 1. Springer, 1997.

[73] K. C. Lee and H.-H. Lee. Network-based fire-detection system via controller area network for smart home automation. *Consumer Electronics, IEEE Transactions on*, 50(4):1093–1100, 2004.

[74] F.-L. Lian, J. R. Moyne, and D. M. Tilbury. Performance evaluation of control networks: Ethernet, controlnet, and devicenet. *Control Systems, IEEE*, 21(1):66–83, 2001.

[75] D. Liu and P. Ning. Efficient distribution of key chain commitments for broadcast authentication in distributed sensor networks. In *Proc. of the 10th Annual Network and Distributed System Security Symposium*, pages 263–276, 2003.

[76] D. Liu and P. Ning. Multilevel μtesla: Broadcast authentication for distributed sensor networks. *ACM Trans. Embed. Comput. Syst.*, 3:800–836, November 2004.

[77] M. Macchetti and P. Rivard. Small-scale variants of the secure hash standard. In *ECRYPT workshop on RFID and lightweight cryptography*, Graz, Austria, July 14-15 2005.

[78] A. Martí, J. Campelo, J. Pardo, R. Ors, and J. Serrano. A distributed control system for citric fruits conservation and maturation based on can and internet networks. In *Industrial Electronics, 2007. ISIE 2007. IEEE International Symposium on*, pages 1899–1904. IEEE, 2007.

[79] S. Mason. Vehicle remote keyless entry systems and engine immobilisers: Do not believe the insurer that this technology is perfect. *Computer Law & Security Review*, 28(2):195–200, 2012.

[80] McAfee. Caution: Malware Ahead. accessed 27 October 2013, [Online]. http://www.mcafee.com/us/resources/reports/rp-caution-malware-ahead.pdf, 2011.

[81] G. McDonald, L. O. Murchu, S. Doherty, and E. Chien. Stuxnet 0.5: The missing link. *Symantec Security Response, February*, 26, 2013.

[82] R. C. Merkle. A digital signature based on a conventional encryption function. In *A Conference on the Theory and Applications of Cryptographic Techniques on Advances in Cryptology*, CRYPTO '87, pages 369–378, London, UK, 1988. Springer-Verlag.

[83] C. Miller and C. Valasek. Adventures in Automotive Networks and Control Units. Technical report, IOActive Labs, 08 2013.

[84] R. Mitchell. *Tutorial: Introducing the XGATE Module to Consumer and Industrial Application Developers, March 2006, [Online]*. Freescale, 2004.

[85] P.-S. Murvay and B. Groza. Performance improvements for SHA-3 finalists by exploiting microcontroller on-chip parallelism. In *Risk and Security of Internet and Systems (CRiSIS), 2011 6th International Conference on*, pages 1–7. IEEE, 2011.

[86] P.-S. Murvay and B. Groza. Performance Evaluation of SHA-2 Standard vs. SHA-3 Finalists on two Freescale Platforms. *International Journal of Secure Software Engineering*, 4, 2013. to be published.

[87] P.-S. Murvay and B. Groza. Source Identification Using Signal Characteristics in Controller Area Networks. In *Accepted for publication in IEEE Signal Processing Letters*, 2014.

[88] M. Muter and N. Asaj. Entropy-based anomaly detection for in-vehicle networks. In *Intelligent Vehicles Symposium (IV), 2011 IEEE*, pages 1110–1115. IEEE, 2011.

[89] M. Müter and F. Freiling. Model-based security evaluation of vehicular networking architectures. In *Networks (ICN), 2010 Ninth International Conference on*, pages 185–193. IEEE, 2010.

[90] M. Muter, A. Groll, and F. C. Freiling. A structured approach to anomaly detection for in-vehicle networks. In *Information Assurance and Security (IAS), 2010 Sixth International Conference on*, pages 92–98. IEEE, 2010.

[91] National Institute of Standards and Technology. FIPS 180-3, Secure Hash Standard, Federal Information Processing Standard (FIPS), Publication 180-3. Technical report, Department of Commerce, Aug. 2008.

[92] D. Nilsson and U. Larson. Secure firmware updates over the air in intelligent vehicles. In *Communications Workshops, 2008. ICC Workshops' 08. IEEE International Conference on*, pages 380–384. IEEE, 2008.

[93] D. Nilsson and U. Larson. Simulated attacks on can buses: vehicle virus. In *Proceedings of the Fifth IASTED International Conference on Communication Systems and Networks*, pages 66–72. ACTA Press, 2008.

[94] D. Nilsson, U. Larson, F. Picasso, and E. Jonsson. A first simulation of attacks in the automotive network communications protocol flexray. In *Proceedings of the International Workshop on Computational Intelligence in Security for Information Systems CISISâ 08*, pages 84–91. Springer, 2009.

[95] D. Noonen, S. Siegel, and P. Maloney. Devicenet tm application protocol. In *Proc. 1st International CAN Conference*, 1994.

[96] S. Okada, N. Torii, K. Itoh, and M. Takenaka. Implementation of Elliptic Curve Cryptographic Coprocessor over GF(2m) on an FPGA. In *Proceedings of the Second International Workshop on Cryptographic Hardware and Embedded Systems*, CHES '00, pages 25–40, London, UK, 2000. Springer-Verlag.

[97] C. Paar, T. Eisenbarth, M. Kasper, T. Kasper, and A. Moradi. Keeloq and side-channel analysis-evolution of an attack. In *Fault Diagnosis and Tolerance in Cryptography (FDTC), 2009 Workshop on*, pages 65–69. IEEE, 2009.

[98] A. Perrig. The BiBa one-time signature and broadcast authentication protocol. In *Proceedings of the Eighth ACM Conference on Computer and Communications Security (CCS-8)*, pages 28–37, Philadelphia PA, USA, 2001.

[99] A. Perrig, R. Canetti, D. Song, and J. D. Tygar. Efficient and secure source authentication for multicast. In *Network and Distributed System Security Symposium, NDSS '01*, pages 35–46, 2001.

[100] A. Perrig, R. Canetti, D. Song, and J. D. Tygar. Spins: Security protocols for sensor networks. In *Seventh Annual ACM International Conference on Mobile Computing and Networks (MobiCom 2001)*, pages 189–199, 2001.

[101] A. Perrig, R. Canetti, J. Tygar, and D. X. Song. Efficient authentication and signing of multicast streams over lossy channels. In *IEEE Symposium on Security and Privacy*, pages 56–73, 2000.

[102] S. Petters, K. Elphinstone, and G. Heiser. Trustworthy real-time systems. *Advances in Real-Time Systems*, page 191, 2012.

[103] M. Reinfrank. Why is automotive software so valuable?: or 5000 lines of code for a cup of gasoline less. In *Proceedings of the 2006 international workshop on Software engineering for automotive systems*, pages 3–4. ACM, 2006.

[104] Renesas Electronics. *SH7205 Group Hardware Manual, Rev. 2.00, March 2010, [Online]. Available: http://documentation.renesas.com/eng/products/mpumcu/rej09b0372_sh7205hm.pdf*, 2010.

[105] L. Reyzin and N. Reyzin. Better than biba: Short one-time signatures with fast signing and verifying. In *Proceedings of the 7th Australian Conference on Information Security and Privacy*, ACISP '02, pages 144–153, London, UK, 2002. Springer-Verlag.

[106] R. Rivest. The MD5 message-digest algorithm. RFC 1321, 1992.

[107] Robert Bosch GmbH. *CAN Specification Version 2.0*, April 1991.

[108] Robert Bosch GmbH. *CAN with Flexible Data-Rate, Specification Version 1.0*, April 2012.

[109] N. Romero-Zurita, D. McLernon, M. Ghogho, and A. Swami. PHY Layer Security Based on Protected Zone and Artificial Noise. *Signal Processing Letters, IEEE*, 20(5):487–490, 2013.

[110] I. Rouf, R. Miller, H. Mustafa, T. Taylor, S. Oh, W. Xu, M. Gruteser, W. Trappe, and I. Seskar. Security and privacy vulnerabilities of in-car wireless networks: A tire pressure monitoring system case study. In *Proceedings of USENIX Security Symposium*, 2010.

[111] Y. Shoukry, P. Martin, P. Tabuada, and M. Srivastava. Non-invasive Spoofing Attacks for Anti-lock Braking Systems. In *Cryptographic Hardware and Embedded Systems - CHES 2013*, volume 8086 of *Lecture Notes in Computer Science*, pages 55–72. Springer Berlin Heidelberg, 2013.

[112] S. A. Shweta, D. P. Mukesh, and B. N. Jagdish. Implementation of controller area network (can) bus (building automation). In *Advances in Computing, Communication and Control*, pages 507–514. Springer, 2011.

[113] F. Simonot-Lion and Y. Trinquet. Vehicle Functional Domains and Their Requirements. In *Automotive Embedded Systems Handbook*, pages 1–22. CRC PRess, 2009.

[114] M. E. Steurer. Multicollision attacks on iterated hash functions. Technical report, TU Graz, July 2006.

[115] G. E. Suh, C. W. O'Donnell, and S. Devadas. AEGIS: A single-chip secure processor. *Information Security Technical Report*, 10(2):63 – 73, 2005.

[116] C. Szilagyi and P. Koopman. Flexible multicast authentication for time-triggered embedded control network applications. In *Dependable Systems & Networks, 2009. DSN'09. IEEE/IFIP International Conference on*, pages 165–174. IEEE, 2009.

[117] C. Szilagyi and P. Koopman. Low cost multicast authentication via validity voting in time-triggered embedded control networks. In *Proceedings of the 5th Workshop on Embedded Systems Security*, page 10. ACM, 2010.

[118] S. Tillich and M. Wójcik. Security analysis of an open car immobilizer protocol stack. In *Trusted Systems*, pages 83–94. Springer, 2012.

[119] M. S. Turan, R. Perlner, L. E. Bassham, W. Burr, D. Chang, S. jen Chang, M. J. Dworkin, J. M. Kelsey, S. Paul, and R. Peralta. Status Report on the Second Round of the SHA-3 Cryptographic Hash Algorithm Competition. NIST Interagency Report 7764, 2011.

[120] A. Van Herrewege, D. Singelee, and I. Verbauwhede. CANAuth-a simple, backward compatible broadcast authentication protocol for CAN bus. In *9-th Embedded Security in Cars Conference*, 2011.

[121] R. Verdult, F. D. Garcia, and J. Balasch. Gone in 360 seconds: Hijacking with hitag2. In *Proceedings of the 21st USENIX conference on Security symposium*, pages 37–37. USENIX Association, 2012.

[122] X. Wang and H. Yu. How to break MD5 and other hash functions. In *Advances in Cryptology - EUROCRYPT 2005*, volume 3494 of *LNCS*, pages 561–561. Springer Berlin / Heidelberg, 2005.

[123] C. Wenzel-Benner and J. Gräf. SHA3 Results - XBX:eXternal Benchmarking eXtension. accessed 29 December 2012, [Online]. Available: http://xbx.daslabor.org/trac/wiki/Sha3Results, 2012.

[124] M. Wolf and T. Gendrullis. Design, implementation, and evaluation of a vehicular hardware security module. *Information Security and Cryptology-ICISC 2011*, pages 302–318, 2012.

[125] M. Wolf, A. Weimerskirch, and C. Paar. Secure in-vehicle communication. In *Embedded Security in Cars Securing Current and Future Automotive IT Applications*. Springer Verlag, 2006.

[126] M. Wolf, A. Weimerskirch, and T. Wollinger. State of the art: Embedding security in vehicles. *EURASIP Journal on Embedded Systems*, 2007, 2007.

[127] H. Wu. The Hash Function JH. Submission to NIST (round 3), 2011.

[128] T. Xia, C. Baird, G. Jallo, K. Hayes, N. Nakajima, N. Hata, and P. Kazanzides. An integrated system for planning, navigation and robotic assistance for skull base surgery. *The International Journal of Medical Robotics and Computer Assisted Surgery*, 4(4):321–330, 2008.

[129] B. Xiao, B. Yu, and C. Gao. Detection and localization of sybil nodes in vanets. In *International Conference on Mobile Computing and Networking: Proceedings of the 2006 workshop on Dependability issues in wireless ad hoc networks and sensor networks*, volume 26, pages 1–8. Citeseer, 2006.

[130] T. Ziermann, S. Wildermann, and J. Teich. Can+: A new backward-compatible controller area network (can) protocol with up to 16× higher data rates. In *Design, Automation & Test in Europe Conference & Exhibition, 2009. DATE'09.*, pages 1088–1093. IEEE, 2009.