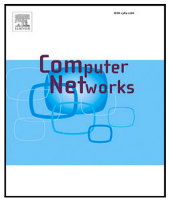
Computer Networks 218 (2022) 109377

Contents lists available at ScienceDirect

Computer Networks

journal homepage: www.elsevier.com/locate/comnet



Designing and implementing an AUTOSAR-based Basic Software Module for enhanced security

Giampaolo Bella [a], Pietro Biondi [a,*], Gianpiero Costantino [b], Ilaria Matteucci [b]

[a] Dipartimento di Matematica e Informatica Università degli Studi di Catania, Catania, Italy
[b] Istituto di Informatica e Telematica, Consiglio Nazionale delle Ricerche, Pisa, Italy

ARTICLE INFO

Keywords:
Onboard Communication
Automotive
Cybersecurity
Confidentiality

ABSTRACT

Electronic Control Units (ECUs) communicate with each other to accomplish the functionalities of modern vehicles. ECUs form an in-vehicle network that is precisely regulated and must be adequately protected from malicious activity, which has had several outbreaks in recent years. Therefore, we present CINNAMON, an AUTOSAR-based Basic Software Module that aims at confidentiality, integrity and authentication, all at the same time, for the traffic exchanged over the bus protocols that AUTOSAR supports. CINNAMON in fact stands for Confidential, INtegral aNd Authentic onboard coMmunicatiON.

This article introduces the requirements and specification of CINNAMON in a differential fashion with respect to the existing *Secure Onboard Communication* Basic Software Module, which does not include confidentiality. As a result, CINNAMON exceeds SecOC at least against information gathering attacks. The article then defines three security profiles, regulating also the freshness attribute appropriately. Most importantly, CINNAMON is not a simple academic exercise because it is implemented in a laboratory environment on commercial ECUs, thus reaching the level of TRL 4, "Component and/or breadboard validation in laboratory environment". The runtimes obtained on inexpensive devices are reassuring, paving the way for a possible large-scale application.

1. Introduction

Modern vehicles embed so much digital technology that they resemble a network of computers. Each car, in particular, hosts several Electronic Control Units (ECUs), which need to exchange a huge amount of data for the various functions of the car to work smoothly. Components such as air-bags, power doors and the advanced driver-assistance systems (ADAS), need to communicate to ensure the synergistic functioning of all. To communicate one another, ECUs may adopt several buses, such as the Controller Area Network (CAN) [1], FlexRay [2], Ethernet [3].

Cars may also handle driver's personal data such as driving style [4], location history [5] and even more general data such as cabin preferences, music preferences and credit card details [6]. Each type of data normally travels on a separate network: for example, while the driving style can be gathered from data transported on the frame bus, music and other similar preferences and are normally transported on the infotainment network, which is often isolated through a security gateway. In consequence, the in-vehicle network should be designed and partitioned with care, enforcing well-defined security policies in each subnetwork much the same way as it is routinely done in an institutional network.

However, experience shows that in-vehicle networks are not always hardened appropriately. The most popular example is the attack of 2015 by Miller and Valasek to drive a Jeep Cherokee off track [7]. The researchers first reverse-engineered the target ECUs to understand what functionality was associated to what frame, then flashed the control units to control them and ultimately send arbitrary frames. Similarly, hackers exploited a vulnerability of the infotainment system of a General Motors car to steal data from the infotainment system via the Internet [8]. In 2016, researchers discovered a number of vulnerabilities that, when combined, allowed them to remotely violate a Tesla Model S [9]. In 2018, the Keen Security Lab presented a set of vulnerabilities on BMW cars enabling an attacker to inject Unified Diagnostic Services (UDS) frames into the CAN network bypassing the central gateway [10]. Another relevant example is a 2020 attack that exploited a Bluetooth vulnerability on a Toyota Lexus and then continued with frame reverse engineering [11].

* Corresponding author.
E-mail addresses: giamp@dmi.unict.it (G. Bella), pietro.biondi@phd.unict.it (P. Biondi), gianpiero.costantino@iit.cnr.it (G. Costantino), ilaria.matteucci@iit.cnr.it (I. Matteucci).

https://doi.org/10.1016/j.comnet.2022.109377
Received 3 September 2020; Received in revised form 11 April 2022; Accepted 19 September 2022
Available online 23 September 2022

It is clear that exploiting such vulnerabilities always required the attacker to reverse engineer the association of a frame with a specific signal for an ECU hence with a precise functionality [12]. Therefore, the attacks leverage the lack of confidentiality for data in transit within the vehicle network, and such a lack permits information gathering. We therefore contend that confidentiality ought to be added *by design* to in-vehicle communications, and we set out to reach it by means of encryption. Encryption is the *de facto* standard to protect data from disclosure while in transit. Many examples confirm this claim, such as client–server communications over HTTPS and popular chat services pervasively exposed via mobile apps. Encryption is also one of the measures that (art. 32 of) the General Data Protection Regulation advocates to protect personal data in any application scenario that treats such data [13], and driver profiling is a risk [4,5]. Additional motivation to our work comes from Addendum 154 to UN Regulation 155, which postulates that "*Confidential data transmitted to or from the vehicle shall be protected*" [14].

AUTOSAR is a partnership of industries aimed at open and standardised software architectures for ECUs. It provides a Classic Platform, which is a Software Platform defined for deeply embedded systems and Application Software with high demands regarding predictability, safety and responsiveness. Regarding security, AUTOSAR defines the *Secure Onboard Communication* BSW module, named

SecOC, through its requirements [15] and specifications [16]. In short, SecOC aims at integrity of onboard communications and authentication of ECUs that act as senders; by contrast, it does not consider confidentiality. This article presents in full detail the CINNAMON Basic Software module (for brevity, CINNAMON module in the following), an

AUTOSAR-based module that exceeds SecOC by providing also encryption over the communication bus.

CINNAMON rests on our conference paper [17] and extends it in (various directions) as follows:

1. The requirements and the specification of CINNAMON are generalised and no longer limited to the CAN bus.
2. The definitions of the three security profiles are largely extended and clarified.
3. The implementations of Message Authentication Code (MAC) and of encryption are combined on all three (rather than on just one) security profiles.
4. The definitions of suitable data structures in support of two different treatments of freshness, known as Single Counter and Multiple Counter, are provided and made available.
5. The full prototype implementations of all three (rather than just one) security profiles on CAN bus are reached on STM32 boards resembling automotive ECUs, then experimented with, producing promising runtimes, and finally publicly released.

As a result, the present manuscript is almost twice as long as our latest publication [17]; most importantly, the contribution to knowledge developed in this article makes CINNAMON reach the development level of TRL 4, "Component and/or breadboard validation in laboratory environment".

The structure of this article is simple. Section 2 introduces the assumed threat model. Section 3 presents the requirements of CINNAMON and Section 4 its specification and integration in AUTOSAR Classic Platform. Section 5 details the CINNAMON Security profiles. Sections 6 and 7 discuss the implementation of the CINNAMON architecture prototype and CINNAMON security profiles, respectively. Section 8 shows the runtimes obtained through our experiments. Section 9 treats the related work. Section 10 draws conclusions.

2. Threat model

This paper assumes a threat model with an active attacker who may exploit some vulnerabilities of in-vehicle network to gain some digital access to the car, either locally or remotely. More precisely, our attacker attempts to carry out two sets of malicious activities:

- **Malicious activity set 1**, the injection of frames she altered or built from scratch using known data, thus manipulating the data processed by the target ECUs to trigger specific functionalities. This activity set includes:
  - *tampering*, the manipulation of frames with the aim of invalidating their contents so that receiving ECUs cannot perform execute the operations that were originally meant;
  - *fuzzing*, the manipulation of frames with the aim of studying the behaviour of target ECUs;
  - *forging*, the generation of a valid frame with the aim of generating a valid signal and activating a specific ECU functionality;
  - *replaying*, the reuse of valid frames with the aim of repeating the generation of a valid signal and reactivating a specific ECU functionality.
  - *masquerading*, the generation of a valid frame with the aim of abusing the identifier of another, genuine ECU.
- **Malicious activity set 2**, the collection of information about the running protocols and other mechanisms in place in the network she observes, in particular acquire exchanged data. This activity set includes:
  - *sniffing*, the capture of frames in transit with the aim of learning the data they carry;
  - *information gathering*, the capture of frames in transit with the aim of identifying and interpreting the full set of data they carry, such as payloads and their associated ECU functionalities.

The attacker's malicious activities are clearly related to the security properties and attributes that we would like to establish, in the sense that each set of malicious activities attempts to undermine a set of security properties and, in turn, that set of security properties attempts to thwart the given set of malicious activities. In particular:

- Malicious activity set 1 relates to security property and attributes:
  - *authentication*, the identity of the sender of a given frame can be verified;
  - *integrity*, the content of a given frame is not altered during transmission;
  - *freshness*, it can be verified whether a given frame was already received.
- Malicious activity set 2 relates to security property set:
  - *confidentiality*, the content of a given frame is not disclosed to unauthorised entities.

Our attacker is limited in the sense that she only has partial control of ECUs, hence she cannot:

- obtain privileged access to any ECUs and, in particular,
- access the keys to be used for MACs and cryptographic operations, which are assumed to be deployed on each ECU and protected by a Crypto Service Manager (CSM) or similar solutions.

3. CINNAMON requirements

This section introduces the requirements of CINNAMON and positions the new module within the AUTOSAR Classic Platform. CINNAMON inherits most of its requirements from AUTOSAR Secure On-Board Communication module (SecOC) without modifications, hence those are not discussed here. By contrast, CINNAMON inherits and modifies some requirements from SecOC in order to account for encryption. Such new requirements are presented below with the caveat that the modifications determined by the new module are highlighted in boldface; also, the new requirements coherently preserve the numbering of the old ones.

Table 1
CINNAMON_00003 requirement.

Functional Requirement	Configuration of different security properties/requirements
Type:	Valid
Description:	Different security properties**, notably including confidentiality,** shall be configurable.
Rationale:	The assessment may vary in several parameters and its security needs. Thus the level of protection shall be configurable to adapt to these needs by means of a set of adequate parameters.
Use case:	Security experts define the different security properties. For every message with security protection needs, the appropriate properties may be selected.
Corresponding SecOC Requirement	SRS_SecOC_00003

Table 2
CINNAMON_00005 requirement.

Initialisation	Initialisation of security information
Type:	Valid
Description:	The CINNAMON module's security configuration shall get initialised at module start-up.
Rationale:	The CINNAMON module needs security configuration information (Key-IDs **for calculating MACs**, Freshness Values, **encryption initialisation parameters and Key-IDs for encryption**) to perform its operations. Therefore, this information shall get recovered and configured before it starts its processing operation.
Use case:	CINNAMON loads the ID of the messages, the authorised authentication retry counter and the properties, **including confidentiality**, that are used for the processing of its incoming communications from upper and lower layers.
Corresponding SecOC Requirement	SRS_SecOC_00005

3.1. Functional requirements

Requirement CINNAMON_00003 (Table 1) configures different security properties, with the noteworthy inclusion of confidentiality. In, essence, it prescribes that security experts be able to define the level of protection of onboard communication messages and the parameters needed to configure the functionalities of the module, including encryption and decryption.

3.2. Initialisation requirements

Requirement CINNAMON_00005 (Table 2) initialises the security information and also covers encryption. In fact, the requirements explicitly refers to encryption initialisation parameters and keys needed for the encryption phase. Such phase aims at including confidentiality among the set of CINNAMON security properties.

Requirement CINNAMON_00030 (Table 3) inherits from SecOC that it shall be possible to extract the payload from secured messages without authentication and insists on this to be possible even if the payload is encrypted.

3.3. Non-functional requirements

Requirement CINNAMON_00025 (Table 4) refers to performance, thus it regulates the computation time to perform security operations, in particular for dealing with cipher-texts.

Table 3
CINNAMON_00030 requirement.

Normal Operations	Support of capability to extract Authentic PDU without Authentication
Type:	Valid
Description:	CINNAMON shall be capable to extract the payload from Secured frames, without Authentication, even if the payload **is encrypted**.
Rationale:	CINNAMON can be used as an extractor of payload from Secured frames, to enable low latency GW behaviour when a part of downstream communication clusters does not require authentication of frames.
Use case:	Gateway.
Corresponding SecOC Requirement	SRS_SecOC_00030

Table 4
CINNAMON_00025 requirement.

Non-functional requirements (Timing)	Authentication and verification processing time
Type:	Valid
Description:	Authentication, verification, **encryption and decryption** processing shall be performed in a timely fashion so that the real time critical signals do not get affected.
Rationale:	Transmission and reception of the time critical message between the running applications of two or more peers shall not get penalised by the additional processing of their underlying communication software layers such that the signals are finally rejected. It is necessary that when time critical messages are transmitted and received through secured messages, the additional processing required by CINNAMON remains under a value that is predictable and compatible with the time constraints of the concerned signals.
Use case:	A legitimate **encrypted and** authenticated message is **decrypted,** verified and passed to the receiving ECU within the expected time-frame without experiencing signal monitoring errors.
Corresponding SecOC Requirement	SRS_SecOC_00025

4. CINNAMON specification

The CINNAMON module is a Basic Software (BSW) module capable of protecting on-board CAN Bus communications and is based on AUTOSAR in the sense that it extends AUTOSAR in terms of possible functionalities provided and can be integrated into the current AUTOSAR architecture. Moreover, CINNAMON is designed as a single module that is built to provide security functionalities within the AUTOSAR framework. Thus, it is designed and implemented (Sections 6 and 7) to work with the other modules already present into the AUTOSAR Classic Platform.

4.1. Integration with AUTOSAR

CINNAMON is, as SecOC, part of the Communication Services of the AUTOSAR Classic Platform, as depicted in Fig. 1. It encapsulates the SecOC module and inherits its API to interact with the Protocol Data Units (PDU) Router component and with the cryptographic services provided by the *Crypto Service Manager* (CSM). Also, our module interacts with the Run-Time Environment to manage counters and keys. More specifically, the PDU Router module provides services for routing Protocol Data Units between modules such as the communication and transport modules. As for the Crypto Service Manager, it provides

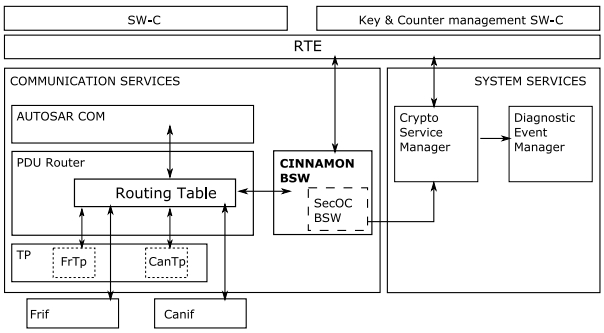


Fig. 1. Integrating the CINNAMON BSW module within AUTOSAR Classic Platform.

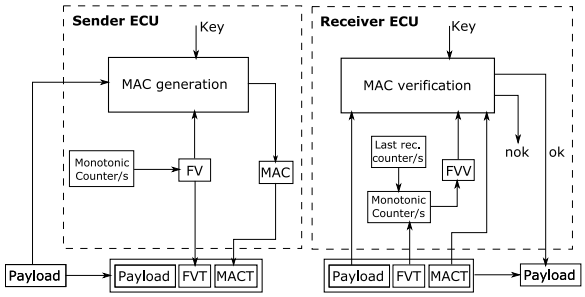


Fig. 2. SecOC MAC generation and verification [16].

services to allow single access to basic cryptographic functionality for all software modules. Hence, CSM provides a standardised interface at the software levels to access these features. Since CINNAMON is based on AUTOSAR, it is able to use other AUTOSAR components to carry out communication and security operations.

CINNAMON acts as a middle-layer between the low-layer communication module, i.e., TP, and the upper layer software module, i.e., AUTOSAR COM. In addition, our module internally manages the communication with the lower level to build and send the secured data using a single PDU. Differently, the last version of SecOC specification [16] suggests to use two PDU, one dedicated to store information used to authenticate the sender of the frame, and another one containing the secured frame.

4.2. Authentication and integrity

CINNAMON inherits SecOC authentication and integrity mechanisms, reviewed in Fig. 2.

AUTOSAR assumes that all ECUs have the cryptographic keys to handle Message Authentication Codes (MACs) (see [18]). Moreover, an external Freshness Value Manager provides counters to both sender and receiver to support the freshness of exchanged frames.

CINNAMON inherits the same prerequisites, briefly recalled here. Let us consider a sender ECU and a receiver ECU. Before sending a payload, the sender generates the MAC starting from the payload and possibly the *Freshness Value* calculated according to the Monotonic Counter (Fig. 2) provided by the Freshness Value Manager (an ECU may decide to ignore the Freshness Value). So, the secured frame is composed by the payload, the truncated MAC (MACT in Fig. 2) and, optionally, the truncated freshness value (FVT).

The receiver has to validate the frame before accepting it and does this by verifying the MAC. In fact, the receiver generates a freshness value for verification (FVV) starting from the Monotonic Counter (Fig. 2) received by the Freshness Value Manager and the previously received freshness value (the latest received counter in Fig. 2). Then, it calculates the MAC by using the received payload and the FVV. If the outcome equals the received MACT, then the payload is accepted, otherwise it is discarded.

4.3. Confidentiality

While MAC operations are normally fast, hence not problematic on inexpensive ECUs, it can be anticipated that the implementation of encryption and decryption primitives may cause a computational bottleneck. In consequence, the choice of the cryptographic scheme will have to be made with care.

Also, this specification purposely is not prescriptive with respect to the choice between the encrypt-then-MAC or MAC-then-encrypt approaches. This will be made at implementation level, depending on the adopted cryptographic scheme, its features and possible vulnerabilities. Another factor for this choice will be the specific transport protocol of the application scenario.

4.4. Freshness

The freshness value refers to a Monotonic Counter (MC), termed Freshness Counter (FC), which is used to guarantee the freshness of communication and must be provided by a Freshness Value Manager (FVM) unit which regularly distributes the freshness values on the network. In addition, the FVM takes care of synchronising and updating the freshness values in the ECUs appropriately. The FVM also allows the FC to be refreshed in a way that mitigates possible attacks such as replay and MiTM.

CINNAMON implements the freshness mechanism using two different ways that follows the AUTOSAR specifications. Note that, in total AUTOSAR describes three different approaches to building the freshness counter, two based on counters (single or multiple) and one on timestamps.[1]

In the approach based on *Single Freshness Counter*, the FVM supplies the FV to nodes connected to the network and increases the counter each time a frame is sent in the communication channel. To guarantee freshness, the FCs of the sender and receiver should be increased synchronously. Thus, the FC must be incremented for each outgoing frame that has been validated on the receiving side.

The approach based on *Multiple Freshness Counters* uses four counters: *Trip Counter*, *Reset Counter*, *Message Counter* and *Reset Flag*. The FVM manages two of them, the Trip and Reset Counter. These counters are incremented according to specified criteria [16]. For example, the Reset Counter is incremented by 1 at regular time intervals. The multiple freshness counter approach requires the use of a `ClearAcceptanceWindow` parameter that represents an admissible freshness interval to counter potential lack of synchronisation.

5. CINNAMON security profiles

As in the Secure On Board Communication module, also in the CINNAMON module it is possible to define and manage various security profiles.

Security Profiles provide a consistent set of values for a subset of configuration parameters that are relevant for the configuration of CINNAMON. This is in line with the Secure Onboard Communication module [16]. A CINNAMON Security Profile is defined as the configuration of the following mandatory parameters.

- `algorithmFamily:String [0..1]` is the first parameter that characterises the used authentication algorithm. This parameter identifies the family of authentication algorithms.
- `algorithmMode:String [0..1]` is the second parameter that characterises the used authentication algorithm. This parameter identifies which MAC algorithm of the family is used.

[1] In this article, we do not consider the FV based on timestamp due to hardware constrains.

Table 5
CINNAMON Security Profile 1.

Parameter	Configuration value
`algorithmFamily`	Chaskey
`algorithmMode`	Chaskey_MAC
`algorithmSecondaryFamily`	Not set
`SecOCFreshnessValueLength`	Not set
`SecOCFreshnessValueTruncLength`	Not set
`SecOCAuthInfoTruncLength`	24 bit
`algorithmFreshnessValue`	Not set
`algorithmEncryption`	SPECK64/128

- `algorithmSecondaryFamily:String [0..1]` is the third parameter that characterises the used authentication algorithm. This parameter identifies a secondary family of the chosen algorithm, if any.
- `authInfoTxLength:PositiveInteger` denotes the length of the truncated MAC.
- `freshnessValueLength:PositiveInteger` denotes the length of generated freshness value.
- `freshnessValueTruncLength:PositiveInteger` denotes the length of the truncated freshness value.
- `algorithmFreshnessValue:String [0..1]` denotes the algorithm used to generate the freshness value.
- `algorithmEncryption:String [0..1]` denotes the encryption algorithm.

Note that the first six parameters are inherited from SecOC, while the last two are typical of CINNAMON. This paper defines three example security profiles.

5.1. Security Profile 1

The CINNAMON Security Profile 1 is in Table 5. It does not set any parameters related to the FV and only uses the MAC and encryption algorithm parameters set as:

- Chaskey [19] with `freshnessValueTruncLength` of 24 bits. Chaskey is a Message Authentication Code (MAC) algorithm whose design is thought for 32-bit micro-controller architectures. It is robust under truncation and, since it is employed to produce a tag size of 128 bits, a choice that largely satisfies the official recommendation of exceeding 64 bits, it does not suffer neither of tag guessing (finding counter-images) nor tag collision (birthday attacks).
- SPECK64/128, that is a lightweight block cipher publicly released by the NSA in 2013 [20]. Note that no successful attack on full-round SPECK64/128 (27-rounds) is known. By contrast, reducing the number of rounds invites *differential cryptanalysis attacks* in the standard attack model, which succeed when rounds are reduced to approximately 70%–75%; still, such attacks are only marginally faster than brute-force [21].

5.2. Security Profile 2

Security Profile 2 introduces a freshness value, so it is designed to avoid replay attacks on the communication channel, as shown in Table 6. As before, we define the parameters used by this profile with its respective values. It can be seen that FV is based on Single Counter, that `freshnessValueLength` is of 64 bits and `freshnessValueTruncLength:PositiveInteger` of 8 bits, so FVT is 8 bit long. Finally, the choices of MAC function and the cryptographic scheme remain the same as in Security Profile 1.

Table 6
CINNAMON Security Profile 2.

Parameter	Configuration value
`algorithmFamily`	Chaskey
`algorithmMode`	Chaskey_MAC
`algorithmSecondaryFamily`	Not set
`SecOCFreshnessValueLength`	64 bit
`SecOCFreshnessValueTruncLength`	8 bit
`SecOCAuthInfoTruncLength`	24 bit
`algorithmFreshnessValue`	Single Counter
`algorithmEncryption`	SPECK64/128

Table 7
CINNAMON Security Profile 3.

Parameter	Configuration value
`algorithmFamily`	Chaskey
`algorithmMode`	Chaskey_MAC
`algorithmSecondaryFamily`	Not set
`SecOCFreshnessValueLength`	64 bit
`SecOCFreshnessValueTruncLength`	8 bit
`SecOCAuthInfoTruncLength`	24 bit
`algorithmFreshnessValue`	Multiple Counter
`algorithmEncryption`	SPECK64/128

5.3. Security Profile 3

Security Profile 3 is a modification of the previous one by means of a different method to generate the FV, as shown in Table 7. The method relies on *Multiple Counters* (see Appendix).

Once the FV is generated, the sending ECU calculate the MAC of the payload and truncate the output by taking only 3 bytes. Also, the FV is truncated to 8 bits and assigned to the FVT variable, which is to be inserted in the payload.

On the other side, the receiver must perform the reverse operations. So, it first decrypts the received payload, then executes the freshness value verification algorithm (see Appendix) to obtain a FV of 32 bits starting from the 8 bits of FVT. This operation is needed to check that the counters are synchronised. Once the FV is obtained, the receiver builds its payload to calculate the MAC and carry out the verification.

6. CINNAMON implementations setup

This section describes the CINNAMON prototype implementations by presenting its integration with the AUTOSAR Classic Platform and the implementation choices to fulfil the Security Profiles requirements.

The CAN protocol is designed originally for multiplex electrical wiring within vehicles. Messages are commonly named *CAN frames* and each frame contains various fields. These include an Arbitration field carrying the frame ID, also used for arbitration, a Control field for control signals and a Data field for the payload that spans over up to 64 bits of data, and carries the payload of the frame. The latest version is CAN2.0, dating back to 1991. The 2.0A version carries an 11 bit frame identifier.

The main drawback of the CAN bus protocol is that it has been designed without security on top of it. Hence, in order to provide our prototype implementation of CINNAMON, we use the CAN protocol as example of a real protocol completely unsecured but widely used in the automotive domain.

6.1. Testbed resembling AUTOSAR classic platform

In order to resembling the architecture of the AUTOSAR Classic Platform, we set up our testbed consisting of two ECUs and a laptop interconnected via CAN bus. The laptop can send and receive frames in the channel, which can in turn be connected to other networks, for example also through the PDU Router, as in modern automotive networks. Then, the other two are on STM32F407 Discovery boards

connected to the laptop through a USB-to-CAN device. The boards come with an ARM Cortex M4 processor each, physical input buttons and light emitting diodes (LED) for visual outputs. Additionally, the boards are equipped with an additional STM32F4 DISCOVERYCOMM shield to provide CAN bus connectivity.

The laptop acts as FVM in the experiments that are explained below, and it is assumed that all participating ECUs are provided with the necessary cryptographic keys. We can therefore conjecture that all cryptographic operations require interaction with the Crypto Service Manager. The laptop runs software *IXXAT canAnalyser 3* [22] to manage the USB-to-CAN device, while the two boards run our code discussed in the sequel of this manuscript. Moreover, we can input to canAnalyser 3 specific frames that we want the laptop to send, and this is useful to test broadcast reception, namely by both boards in our setup.

The laptop may also send manually crafted frames so that we can check broadcast reception, namely by both boards in our testbed.

6.2. Implementation complexities

Our initial choice to use Chaskey as a MAC function upon the basis of its specifications was a lucky one. The function was reasonably easy to implement and appreciably fast since the initial experiments. Tags were truncated to 24 bits.

However, the encryption algorithm had to be chosen with care. Our obvious, initial candidate was AES but it produced a data field of at least 128 bits, while we aimed at a data field of 64 bits only so that it could be accommodated in just one frame. On the other hand, a 64 bit version of AES would be weaker and is not standardised. We also experimented with DES, 3DES and Blowfish, but their main drawback for our application was the computational overhead. By contrast, SPECK64/128 [20] uses a 128 bit key, produces a 64 bit output and is lightweight, so it turns out the optimal candidate here.

Further complexity derived from the implementation of the FVM, which, of course, behaves differently from all other boards. We had to decide whether and how to simulate it or whether to program it specifically as with the other boards. While our original impulse was towards the first route, we soon found a convenient way to take the second, as we shall see.

Our implementations are publicly available under MIT license [23] so they are fully reproducible.

6.3. Implementation choices

In light of what we just discussed, our implementations adopt SPECK64/128 [20] to encrypt and decrypt CAN frames, Chaskey to calculate MACs and the two FV mechanisms introduced above at specification level. A CINNAMON secured CAN frame is formed by reducing the dimension of the payload. Then, a freshness value is used to guarantee that the frame content is fresh. To complete the data field, an additional block is used for the Message Authentication Code (MAC), which ensures authentication and integrity. Finally, the entire 64 bits of the payload are encrypted to ensure confidentiality. Therefore, our implementations of CINNAMON and its current profiles on CAN bus take the MAC-then-Encrypt approach, as specified in Fig. 3. In operational terms, the sender ECU extracts a key, Key_m, from the CSM and uses it to generate the MAC of payload and FVT; it then truncates the MAC as MACT, extracts another key, Key_e from the CSM to encrypt payload, FVT and MACT. The receiver ECU extracts its copy of the key Key_e (SPECK implements symmetric cryptography) and uses it to decrypt the data field; it then selects payload and FVT, extracts its copy of the key Key_m to re-compute their MAC, hence it truncates that MAC and checks its correspondence with the received MACT.

We are aware that, in general, depending on the chosen algorithm and on the length of the frames, the MAC-then-encrypt approach may turn out less secure than the encrypt-then-MAC approach due to message padding, which may allow an attacker to break the security of the message rebuilding [24]. However, this risk is zeroed in our case because there is no padding needed due to the fixed length of the considered messages.

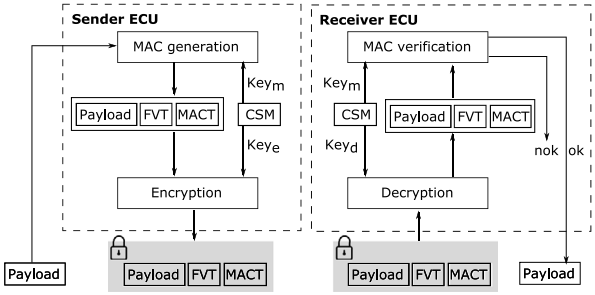


Fig. 3. Complying with CINNAMON on CAN bus.

There is a second reason in support of our choice. The MAC-then-encrypt approach encrypts 64-bit long frames (using encryption algorithms with 64-bit block size and no need for padding). By contrast, using the encrypt-then-MAC approach according to the AUTOSAR specification, the payload (or payload plus FVT) is shorter than 64 bits hence padding would be needed. Most importantly, in frames where the 64 bits are already taken, adding a MAC would necessarily require the transmission of an additional frame to contain it [25].

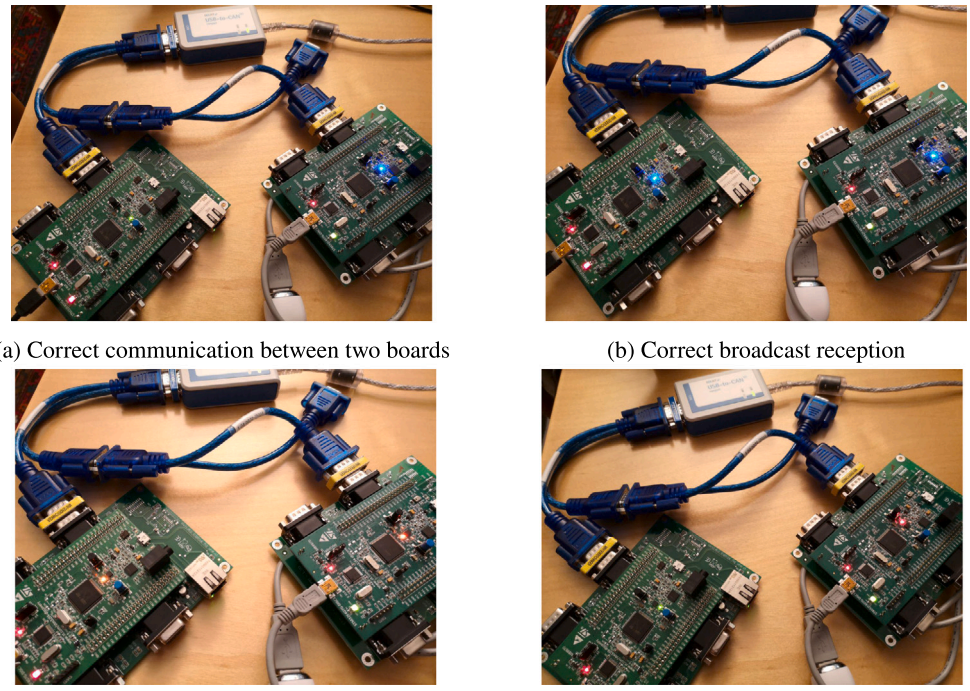
Another effective choice made through our implementations is about the FVM. We opt to simulate it through IXXAT canAnalyser 3 [22], which works well off-the-shelf on our laptop to forward CAN frames back and forth between the two boards. However, it does not provide adequate management of the FC in Single Counter style, namely for Security Profile 2. In the homologous SecOC profile, AUTOSAR adopts a centralised management, precisely by the FVM, of the FC for all frames. On one hand, it is convenient to just use canAnalyser 3 as is but, still, the management of the FC has to be implemented from scratch. So, we take a *distributed state matrix* approach to enable each ECU to keep track of the FC value associated with each frame ID. Therefore, each ECU implements a state matrix to store the FC currently associated to each of the frame IDs the ECU can handle.

6.4. Visual cues

Once CINNAMON is deployed on our boards, we can observe the visual cues from its leds to get a confirmation of the operations that the boards performed. In particular, Fig. 4 shows four notable scenarios. The red led can be noticed on the left hand side of all boards, near the Mini B power connector, indicating operation.

- Fig. 4(a) may refer to all security profiles. The left board sent a frame successfully hence turned on the green led. The right board receives that frame successfully hence turns on the blue led.
- Fig. 4(b) may refer to all security profiles. The laptop sent a frame, both boards received it successfully hence turn on their respective blue led.
- Fig. 4(c) may only refer to Security Profile 2 and Security Profile 3. Both boards received updated FVs from the FVM hence turn on their respective orange led.
- Fig. 4(d) may only refer to Security Profile 2 and Security Profile 3. Both boards were originally synchronised, then the left board was reset and, after that, sent a frame successfully hence turned on the green led. The right board is no longer synchronised because the left board was reset, hence the right board discards the received frame and turns on the red led.

These scenarios will be recalled below to demonstrate our implementations of CINNAMON Security profiles.



(a) Correct communication between two boards

(b) Correct broadcast reception

(c) Correct broadcast reception of new freshness values

(d) Synchronisation failure between two boards

Fig. 4. Led colours representing various board states. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

7. CINNAMON Security Profile Implementation

In this section, we provide the description of our prototype implementation of CINNAMON Security Profiles. For each profile we include also some exemplified code snippets to show the implementation of the most relevant functions.

7.1. Security Profile 1 implementation

Our implementation of CINNAMON Security Profile 1 on CAN bus reviews the CAN frame structures as depicted in Fig. 5, with 40 bits for the payload and 24 bits for the MAC. These fields are encrypted coherently with the profile specification and our implementation choices seen above.

7.1.1. Sending function

Once the code (Code 1) is deployed on the ECUs, the experiment starts when we press the blue button on the sending board to trigger the preparation of a CINNAMON secured CAN frame. This computes the MAC and performs encryption, building a new frame with a data field as seen in Fig. 5. The board then sends the frame on the CAN bus to its peer. More specifically, the sender executes the first `memcpy` which takes as input respectively: the `resMAC` variable which will contain the MAC result, the `chas_mac_create` function which takes the frame as input and returns the MAC and finally the third parameter concerns the size of the `resMAC` variable. Next, the second memcpy is performed which is necessary for creating the frame encryption. The first parameter of the memcpy is `result_Enc` which is the variable that will contain the whole encrypted frame. The second parameter is the `speck_Enc` function which takes the frame as input and calculates its encryption and the third parameter concerns the size of the `result_Enc` variable. Finally, the `CAN_send` function is executed to manage the transmission of the frame on the bus. If the function `CAN_send` is successful, then the green LED will light up, otherwise the red one will light up.

Code 1: Code snippet from sender in Security Profile 1

```
memcpy(resMAC, chas_mac_create(frame),
      sizeof(resMAC));
memcpy(result_Enc, speck_Enc(frame),
      sizeof(result_Enc));
if(CAN_send(1, &result_Enc, 0x0F00) == CAN_OK){
  GPIOD->BSRRH = 0xF000;
  GPIOD->BSRRL = 0x1000;
}else{
  GPIOD->BSRRH = 0xF000;
  GPIOD->BSRRL = 0x2000;
}
```

7.1.2. Receiving function

The receiver board performs the reverse operations (Code 2). It decrypts the frame (`speck_Dec` function), executes the `memcpy` function which takes as input the `result` variable which will contain the decoded frame, and uses the 40 bits of the payload to calculate the MAC, truncates it and compares it to the version stored in the last 24 bits of the data field. The outcomes of this experiment can be seen in Fig. 4(a): if the MAC check gives a positive result, the blue LED will light up, otherwise the red one will light up.

Code 2: Code snippet from receiver in Security Profile 1

```
memcpy(result, speck_Dec(msg.data), sizeof(result));
if(chas_mac_can(result)){
    GPIOD->BSRRH = 0xF000;
    GPIOD->BSRRL = 0x8000;
}else{
    GPIOD->BSRRH = 0xF000;
    GPIOD->BSRRL = 0x4000;
}
```

To check that both boards can receive a secured frame, we generate one and assign it to a frame ID accepted by both boards. Then, we

Code 3: Code snippet from sender in Security Profile 2

```
if (isPressed() == 0x01){
 for(i=0; i<stateMatrixLength; i++){
  if (freshMatrix[i][0] == 0x602){
   freshMatrix[i][1] = freshMatrix[i][1]+1;
   tmp_frame[4] = freshMatrix[i][1]; //Freshness Value
   break;
  }
 }
 tmp_frame[0]=0x20; tmp_frame[1]=0x21; tmp_frame[2]=0x22; tmp_frame[3]=0x23; //example and temporary payload for a
     MAC
 memcpy(resMAC,chas_MAC_create(tmp_frame),
 sizeof(resMAC)); //resMAC has the chaskey MAC tag
 for (h=0; h<8;h++){ //concatenation of the payload with the truncated MAC tag
  frame[h] = tmp_frame[h];
   if (h == 5){
    frame[5]=resMAC[0];
    frame[6]=resMAC[1];
    frame[7]=resMAC[2];
    break;
   }
  }
 memcpy(result_Enc,speck_Enc(frame),sizeof(result_Enc)); //Encryption of the frame with SPECK
 for (h=0; h<8;h++) //result of the encryption copied to the data structure for transmission
 can.data[h] = result_Enc[h];
 if(CAN_send(1, &can, 0x0F00) == CAN_OK){
   GPIOD->BSRRH = 0xF000;
   GPIOD->BSRRL = 0x1000;
 }else{
   GPIOD->BSRRH = 0xF000;
   GPIOD->BSRRL = 0x2000;
 }
}
```

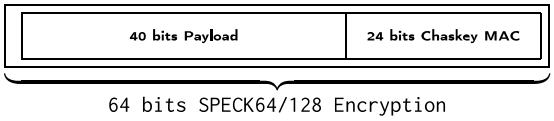


Fig. 5. Data field of a Security Profile 1 frame.

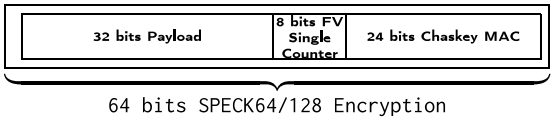


Fig. 6. Data field of a Security Profile 2 frame.

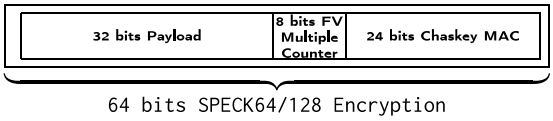


Fig. 7. Data field of a Security Profile 3 frame.

Fig. 8. Structure of counters in multiple freshness counter approach.

manually input such a frame to canAnalyser 3 and both boards receive it successfully hence turn on the blue led (Fig. 4(b)).

7.2. Security Profile 2 implementation

CINNAMON Security Profile 2 extends Security Profile 1 by integrating a FV into the secured frames. Hence the structure of the secured frame generated by Security Profile 2 is the one in Fig. 6: a payload of 32 bits, a FV of 8 bits based on Single Counter and a MAC of 24 bits. Encryption remains the final step prior to sending the frame.

We conventionally use frame ID 602 to carry the FV sent by the FVM.

Each ECU implements a state matrix to store the FC currently associated to each of the frame IDs the ECU can handle, e.g. #10 and #40 in Table 8. The synchronisation among the ECUs is managed by the FVM using the FC value.

7.2.1. Sending function

When a vehicle is powered on, the FVM generates a random value that represents the freshness value FV. Then, the FVM communicates FV to all ECUs, which are all assumed to be able to receive frames from the FVM.

When an ECU wants to send new frames it has to generate them with a certain frame ID. In our testbed, that operations are triggered upon pressing the onboard button (Code 3).

Then, the board increases by one the FV associated with a given ID and assigns it to a specific position of variable `tmp_frame`, the array used for all preliminary operations needed to create the secured frame.

According to the structure of the DBC file, each ECU can handle a specific set of frame IDs and, in turn, each frame ID can be associated with a set of signals. Table 8 provides an example.

Because each ECU stores the last value of the FC that was set for a frame ID, the state matrix of an ECU may look, at some point, as the one in Table 9. When an ECU wants to send payload on a specific

Table 8
Example DBC fragment.

ECU No.	Frame identifier	Signal
#10	201	EngineSpeed OilLevel FuelConsumption ...
	205	RHHighBeamFail LHHighBeamFail ...
	...	...
	205	RHHighBeamFail LHHighBeamFail ...
#40	305	RainSensor LowFuelWarning ...
	...	...
...	...	...

Table 9
Example state matrix for ECU #10.

ECU #10	
Frame identifier	Freshness counter
201	15
205	7
...	...

frame with a certain ID, it queries its state matrix to get the FC value associated with that ID. Then, the ECU increases the retrieved FC by one and assigns it as the new FV for the frame to be sent. At the same time, the ECU updates by one also the FC stored in its state matrix for that frame ID. For example, considering the state matrix of Table 9, when ECU #10 wants to send the frame whose ID is 201, it finds its associated FC of 15, assigns 16 as FV and updates the stored FC as 16.

Then, the board sets a temporary payload to assist the computation of the final payload. Then, the board executes the `chas_MAC_create` function to calculate the corresponding MAC that will be stored in the temporary payload. Moreover, the board truncates the MAC and concatenates it to compose the final payload. After that, the board invokes the cryptographic function `speck_Enc` thereby encrypting the 64 bit payload. Finally, it can be seen that the `CAN_send` API sends the frame just built and the green LED is turned on to signal that all operations have been successfully completed.

7.2.2. Receiving function

On the receiving side, upon reception of a frame, the board decrypts it and stores its contents in the `frame` variable (Code 4).

Code 4: Code snippet from receiver in Security Profile 2

```
memcpy(frame, speck_Dec(can.data), sizeof(frame));
    //the "frame" variable contains the decrypted frame
if(chas_MAC_check(frame)){ //function that verifies
    the MAC, returns 1 if the operation was successful
 for(i=0; i<stateMatrixLength; i++){
  if (stateMatrix[i][0] == can.id &&
      stateMatrix[i][1]+1 == frame[4]){ //check the FV
   stateMatrix[i][1] = stateMatrix[i][1]+1;
   GPIOD->BSRRL = 0x8000; // turn on Blue LED
   break;
  }else
    GPIOD->BSRRL = 0x4000; // turn on Red LED
 }
}
```

Table 10
Example state matrix for ECU #40.

ECU #40	
Identifier	Freshness counter
201	15
305	3
...	...

Then, the board verifies the MAC using the `chas_MAC_check` function, which returns 1 if verification succeeds, 0 otherwise. At this point, the board searches for the frame ID in its own State Matrix to check the frame freshness. For example, let us consider the state matrix illustrated in Table 10.

To validate the FV that comes with a received frame, the receiving ECU compares it to the incremented (by one) version of the FC stored in its state matrix for the ID of the received frame. If they differ, then the ECU rejects the frame and turns on the red led to indicate lack of synchronisation on FV and discards the frame (Fig. 4(d)). Otherwise, it is accepted and then the boards stores the updated FC in its table and switches on the blue led. For example, if ECU #40 receives a frame with ID 201 and FV of 16, and if its state matrix is as in Table 10, then the ECU accepts the frame and updates as 16 the FC stored for the frame. The outcomes of this experiment can be seen, once more, in Fig. 4(a).

If an ECU receives a frame from the FVM, it follows a different procedure. It overwrites its FC values as the frame dictates. This aims at improving synchronicity, and in fact the FVM sents out its frames periodically. For example, if ECU #40 receives a frame from the FVM dictating that the current FV for frame ID 201 is 20, then the ECU updates as 20 the value 15 its example state matrix seen in Table 10.

7.3. Security Profile 3 implementation

This Security Profile manages the FV using the multiple freshness counter approach, so there is a sender-side function to generate the FV and a receiver-side function to reconstruct and verify the FV. Therefore, the data field resembles that of the previous Security Profile with the only difference on the FV counter, as shown in Fig. 7.

The generation and validation algorithms of FV based on Multiple Counter take into account four counters that are increased according to specific conditions provided by AUTOSAR [16]: *Trip Counter*, *Reset Counter*, *Message Counter* and *Reset Flag*, as Fig. 8 shows.

7.3.1. Sending function

The sending function of Security Profile 3 is coded as shown in Code 5. When a new frame is transmitted, the sender ECU executes the `genFreshSender()` function. It takes four variables as inputs: the last *Trip Counter* value received by the FVM, the last *Reset Counter* value received by the FVM, the values sent previously of the *Trip Counter* and *Reset Counter*. The function checks whether the Trip Counter and Reset Counter values received by the FVM are the same as those previously sent. Based on this check, the FV generation algorithm is performed:

- The Trip Counter is increased by one when the FVM starts, resets and when the power status changes.
- The Reset Counter is increased by one at regular time intervals.
- The Message Counter is increased by one value for each message transmission.
- The Reset Flag is represented by the two least significant bits of the Reset Counter.

The resulting FV consists of the Trip Counter, Reset Counter, Message Counter and Reset Flag. Note that, the FVM maintains only the Trip Counter and the Reset Counter that are initialised to one. Instead, the ECU slaves use all four counters which are initialised to zero.

Code 5: Code snippet from generation of the FV multiple counter

```
if( latestTrip == PrevTrip && latestRst == PrevRst ){
        TripCntSender = PrevTrip;
        RstCntSender = PrevRst;
        MsgCntSender = MsgCntSender+1;
        for (i = 7; i >= 0; i--){
            aBitResetFlag[index]=MID(RstCntSender,i,i+1);
            aBitMsgCnt[index]=MID(MsgCntSender,i,i+1);
            index++;
        }
        ResetFlagSender = aBitResetFlag[6]*10 +aBitResetFlag[7];
        FVTrunk = ((aBitMsgCnt[6]*10 +aBitMsgCnt[7])*100)+ResetFlagSender;
        FVTrunk = BinToHex(FVTrunk);
        ResetFlagSender = BinToHex(ResetFlagSender);
}else{
        TripCntSender = latestTrip;
        RstCntSender = latestRst;
        MsgCntSender = 0x01;
        for (i = 7; i >= 0; i--){
            aBitResetFlag[index]=MID(RstCntSender,i,i+1);
            aBitMsgCnt[index]=MID(MsgCntSender,i,i+1);
            index++;
    }
    ResetFlagSender = aBitResetFlag[6]*10 +aBitResetFlag[7];
    FVTrunk = ((aBitMsgCnt[6]*10 +aBitMsgCnt[7])*100)+ResetFlagSender;
    FVTrunk = BinToHex(FVTrunk);
    ResetFlagSender = BinToHex(ResetFlagSender);
}
```

In addition, the aforementioned function also generates the FVT value, which is represented by the last less significant bits of the Message Counter and the Reset Flag. The FVT value will be subsequently inserted in the fifth byte of the frame and will be used by the receiver to build again the FV. Once it is generated, the module builds a frame made up of four bytes of payload and four bytes of FV. At this point, the function to calculate the MAC is invoked. Subsequently, we build a secure frame consisting of four bytes for the payload, the fifth byte is used for the FVT and the remaining three bytes for the truncated MAC. Finally, the entire frame is encrypted with SPECK64/128 and sent into the bus.

7.3.2. *Receiving function*

Upon receiving the frame, the ECU performs the decryption operation and obtains the frame containing the payload, FVT and MAC. The receiver invokes the `genFreshReceiver()` function that takes as input the Trip Counter and Reset Counter parameters received by the FVM and Trip Counter and Reset Counter previously received by the FVM. In addition, the function takes as input the FVT value contained in the fifth byte of the received frame. Hence, the FVV is generated by performing various comparisons on the four counters. All such comparisons may produce 15 different relative conditions among the counters [16], and the value of the receiver's FV is specifically constructed in each case (see Appendix).

Then, a data structure is built to maintain the payload and the FV given as output by the aforementioned function.

We implemented the extraction of the various counters from the FV Multiple counter by means of a function that transforms hexadecimal into an array containing its binary representation. For example if we have the hexadecimal value 87, our method will execute the `BinToHex` function and build the array 1|0|0|0|0|1|1|1.

The MAC is calculated on the data structure to carry out the verification. If the check ends successfully, then the ECU will turn on a blue LED to indicate a positive result, otherwise it will turn on the red LED.

Table 11
Runtimes per primitive.

Algorithm	Time [µs]
Chaskey (MAC)	0.43
SPECK 64/128 (Enc/Dec)	5.36
FV Gen/Ver Single Counter	0.02
FV Gen/Ver Multiple Counter	0.04

To verify that the complex synchronicity demanded in this security profile worked, we successfully run all four experiments whose visual cues are in Fig. 4. In particular, we also cross-checked that, by resetting the sender board hence breaking synchronicity, the receiver board would discard the frame. The outcomes of the latter experiment can be seen in Fig. 4(d).

8. Runtimes

We measured the runtimes of our prototype implementations of CINNAMON and report them here. In our preliminary work [26], we provided a first quantitative evaluation of a CAN-based protocol. In [26], we presented the performance evaluation only of Profile 1, that was designed and implemented in a different way to establish the MAC and frame encryption with respect to what is shown in Section 5.1. Therefore, we improved the CINNAMON design by choosing different encryption algorithms, MAC and freshness and this has allowed us to obtain better computation results related to Profile 1.

All of this leads us to conclude that the additional computations that our module induces only slightly affect the overall performances, despite the fact that our code is only a proof-of-concept and our testbed only contains inexpensive hardware.

Table 11 shows the runtimes of each algorithm in microseconds on a board at 168 MHz. Not surprisingly, encryption and decryption take longer than the other operations but the excess seems acceptable.

It is also interesting to assess the total runtimes of sending or receiving operations for each security profile. We found out that both

Code 6: Code snippet from verification of the FV multiple counter

```
if (latestvalRFlag == rcvRFlag){
      //Format 1 2
      if (LatestTripCntFVM == PrevTrip && LatestRstCntFVM == PrevRst){
      for (i = 7; i >= 0; i--){
          aBitPrevMsgReceiver[index]=MID(PrevMsgCnt,i,i+1);
    index++;
    }
    index = 0;
    PrevRcvUpper = (aBitPrevMsgReceiver[4]*10 +aBitPrevMsgReceiver[5]);
    PrevRcvUpper = BinToHex(PrevRcvUpper);
  PrevRcvlower = (aBitPrevMsgReceiver[6]*10 +aBitPrevMsgReceiver[7]);
  PrevRcvlower = BinToHex(PrevRcvlower);
  rcvLower = (aBitFVTrunk[4]*10 +aBitFVTrunk[5]);
  rcvLower = BinToHex(rcvLower);
  if(PrevRcvlower < rcvLower){ //Format 1
      TripCntReceiver = PrevTrip;
      RstCntReceiver = PrevRst;
      MsgCntReceiver = PrevRcvUpper*100+rcvLower;
      MsgCntReceiver = BinToHex(MsgCntReceiver);
      ResetFlagReceiver = (aBitFVTrunk[6]*10 +aBitFVTrunk[7]);
      ResetFlagReceiver = BinToHex(ResetFlagReceiver);
    }
    else if(PrevRcvlower >= rcvLower){
        TripCntReceiver = PrevTrip;
        RstCntReceiver = PrevRst;
        MsgCntReceiver = PrevMsgCnt+0x01+rcvLower;
        MsgCntReceiver = BinToHex(MsgCntReceiver);
        ResetFlagReceiver = (aBitFVTrunk[6]*10 +aBitFVTrunk[7]);
        ResetFlagReceiver = BinToHex(ResetFlagReceiver);
       }
      }
      //Format 3
      else if (LatestTripCntFVM >= PrevTrip && LatestRstCntFVM >= PrevRst){
          TripCntReceiver = LatestTripCntFVM;
          RstCntReceiver = LatestRstCntFVM;
          MsgCntReceiver = 0x00+(aBitFVTrunk[4]*10 +aBitFVTrunk[5]);
          MsgCntReceiver = BinToHex(MsgCntReceiver);
          ResetFlagReceiver = (aBitFVTrunk[6]*10 +aBitFVTrunk[7]);
          ResetFlagReceiver = BinToHex(ResetFlagReceiver);}
```

operations take the same runtimes for each profile, a non-surprising finding due to the fact that the computational overhead is the same in both cases. Then, Table 12 shows the total runtimes of sending or receiving operations, including frame generation upon sending or frame validation upon receiving.

These runtimes remain unvaried over subsequent executions. It can be concluded that CINNAMON adds less than 6 µs to generate or validate a secured frame in any of its security profiles, a finding that we deem very promising for applications demanding a secure CAN bus.

Table 12
Runtimes per profile.

Profile	Time [µs]
CINNAMON Security Profile 1 (SPECK + Chaskey)	5.79
CINNAMON Security Profile 2 (SPECK + Chaskey + FV Single Counter)	5.81
CINNAMON Security Profile 3 (SPECK + Chaskey + FV Multiple Counter)	5.83

9. Related work

In this section, we discuss novelties and advantages of CINNAMON with respect to the state of the art. The discussion identifies and revolves around the following six features $F_1 \ldots F_6$, which are relevant to the industrial uptake of secure CAN communication:

F1. Standard CAN. This feature holds of a protocol when all fields of the frame, which the protocol defines, conform to size and contents as specified by the CAN standard [35].

F2. Frame rate equal to CAN's. This is true for a protocol that does not need to increase the CAN's frame rate.

F3. Payload size not smaller than CAN's. This holds of a protocol that preserves the standard CAN size of 64 bits for the payload size.

F4. Standard AUTOSAR. This holds of a protocol that conforms to the prescriptions of the latest AUTOSAR standard [16]. Note that profiles were introduced in the AUTOSAR standard only in 2014.

F5. No ECU hardware upgrade. This holds of a protocol when it requires no upgrade to the ECUs that can run the CAN protocol, hence no additional features or computational power are needed for the units.

F6. No infrastructure upgrade. This is similar to the previous feature but concerns the network and the overall infrastructure that supports the protocol. Therefore, it is true for a protocol that executes on the same network that underlies the CAN, without additional, dedicated nodes.

Table 13
Contrastive analysis of CINNAMON w.r.t. the related work.

	CANAuth [27]	MaCAN [28]	LCAP [29]	Libra-CAN [30]	TACAN [31]	CaCAN [32]	LeiA [33]	CANcrypt [34]	CINNAMON
F1. Standard CAN	✗	✗	✓	✗	✓	✓	✓	✓	✓
F2. Frame rate equal to CAN's.	✗	✗	✗	✗	✓	✗	✗	✓	✓
F3. Payload size not smaller than CAN's.	✗	✗	✗	✗	✓	✗	✓	✗	✗
F4. Standard AUTOSAR	✗	✗	✗	✗	✗	✗	✓	✗	✓
F5. No ECU hardware upgrade	✗	✗	✓	✗	✗	✓	✓	✓	✓
F6. No infrastructure upgrade	✓	✗	✓	✓	✗	✗	✓	✓	✓
	1	0	3	1	3	2	5	4	5

Table 14
Construction of freshness value for transmission [16].

Trip Counter and Reset Counter comparison	Construction of freshness value for transmission			
	Trip Counter	Reset Counter	Message Counter	Reset Flag
Latest value = Previously sent value	Previously sent value	Previously sent value	Previously sent value +1	The value from the lower end of the reset counter (previously sent value)
Latest value ≠ Previously sent value	Latest value	Latest value	Initial Value +1	The value from the lower end of the reset counter (latest value)

Since the current version of SecOC module has been introduced in 2014 and slightly revised and improved till 2021, the existing solutions to secure the CAN bus can be naturally divided into ante and post 2014.

Among the ante 2014 solutions there are:

- CANAuth [27], 2011. It is based on CAN + [36], which is an extension of the basic CAN protocol in which the data rate is extended in such a way that more bytes can be sent (up to 16 CAN + bytes for each CAN byte) in the same frame. The drawback of this protocol is that it requires to change the transceivers, which must be more powerful to manage the CAN + data rate. This implies that using CANAuth has an impact on hardware, which must be upgraded.
- LCAP [29], 2012, aimed to guarantee message authentication, resistance to replay attacks, and backward compatibility at the same time. It is based on some out-of-band protocol like CAN + . The main drawback is the use of broadcast-based authentication, which increases the traffic in a way directly proportional to the number of nodes in the network.
- MaCAN [28], 2012, is a centralised authentication protocol based on broadcast-based authentication, so it requires CAN + or CAN FD. However, the same protocol was found to be flawed [37].
- Libra-CAN by Groza et al. [30], 2012, a protocol based on a MAC calculated using MD5. Its main drawbacks are high bandwidth and the introduction of hardware capable of understanding and manage the new frame format: Libra-CAN protocol is based on CAN + instead of on CAN.

Referring to the AUTOSAR standard, all solutions listed above do not appear to be based on the requirements and guidelines described in the SecOC module requirements and specification. Moreover, most of them require to redesign the vehicle network architecture to or introduce new nodes or upgrade the ECUs to manage new protocols. The solutions proposed after 2014 are the following ones:

- TACAN [31] shares a master key between an ECU and the *Monitor Node* to generate shared session keys. These are assumed to be stored in a Trusted Platform Module (TPM) [38]. Each ECU embeds unique authentication frames into CAN frames and continuously transmits them through covert channels, which can be received and verified by the Monitor Node. TACAN aims at mitigating suspension, injection and masquerade attack. TACAN does not address confidentiality.
- CaCAN [32] introduces a key distribution phase. Hence, the protocol needs a new component in the architecture to act as a monitoring node. Frames are not sent in broadcast but on a peer-to-peer basis. The protocol is simulated but not implemented on micro-controllers.
- LeiA [33] uses MAC to authenticate messages: for each message, the protocol sends a message in plaintext and another one with the MAC of the message. LeiA rests on a 29-bit message identifier, which is coherent with CAN 2.0B [35].
- CANcrypt [34], 2017, is closely related to our work but does not follow AUTOSAR guidelines. Also TLS-based approaches are valid but demand extra-vehicular Internet connectivity and are limited to time-critical applications due to performance overhead [39].

All these protocols present pros and cons. Table 13 represents a contrastive analysis of the main entries in the related work with respect to all six features. Notably, no protocol ticks all features, but LeiA and CINNAMON are the only ones that are both CAN compliant, based on the AUTOSAR guidelines and, at the same time, require no upgrade to each ECU, or network augmentation with additional components. CANcrypt does not strictly follow all AUTOSAR profiles. In fact, it does not implement any freshness values algorithm able to mitigate the replay attack. Moreover, LeiA and CINNAMON have alternate features F2 and F3. While LeiA keeps the CAN payload size of 64 bits, it doubles each frame, a feature that may produce some safety concerns, as discussed elsewhere [40]. On the contrary, both CINNAMON and CANCrypt satisfy F2 but not F3. Both protocols rely on the CAN frame size of 64 bit. The main difference is that CANcrypt is designed to be applied to only a subset of messages in a quite small network due to the high introduced overhead. Contrarily to most of the other protocols, such as CaCAN, we implement CINNAMON on micro-controller boards resembling the real behaviour and computational power of ECU. The obtained performances are very promising because the new modules does not introduce additional overhead on the communication bus.

Moreover, recent work analyses the impact of introducing security over functional properties of vehicles. Dariz et al. [40,41] presented a trade-off analysis between security and safety when a security solution based on encryption is applied on CAN messages. The analysis is presented considering different attacker models, packet fragmentation issues and the residual probability of error of the combined scheme. Also Groza et al. [42] and Stabili et al. [43] targeted the delicate relation between security and safety. Also, a framework for the specification and automatic generation of security features for communications among AUTOSAR-compliant components must be mentioned [44]. It allows

AUTOSAR designers to add security specifications to the communication model through a dedicated software tool. However, it has not yet been practically used to advance new components or protocols that would combine confidentiality with authentication and integrity.

10. Conclusions

What digital technologies are going to support the functioning and exposed services of the cars we will be driving in the future?

The features and internal diversity of such functioning and services cannot be overestimated. However, it is our belief that they will need to withstand malicious activities hence will need to achieve adequate non-functional properties to thwart those activities.

We began our works (and then opened this article) by arguing that confidentiality is an essential property also in the automotive domain, and that the use of the technical security measure to achieve it, encryption, already is pervasive and also in line with the current European regulation on data treatment. So, such a use of encryption should be extended over the automotive domain, and the recent Addendum 154 to UN Regulation 155 underlines this [14].

Therefore, this article presented CINNAMON, an AUTOSAR-based basic software module aims at confidentiality, integrity and authentication combined together
and, at the same time, enhanced with the freshness attribute. In particular, the introduction of confidentiality makes reverse engineering operations of an attacker harder. For example, understanding communications would require brute-forcing cipher-texts, a daunting task that would, in turn, require the gathering of several pieces of information, such as encryption key, encryption algorithm and general frame semantics. It is reassuring that, because all these activities are unlikely to succeed, the attacker will be unable to forge valid frames and practically abuse real cars the way the news have reported so far.

CINNAMON is general and we have reached a TRL 4 prototype implementation on CAN bus that complies with it. The observed runtimes are promising and only negligibly increase those of SecOC, which does not use encryption. This result supports the claim that securing in-vehicle communications cryptographically is currently possible hence viable for large-scale deployment.

CRediT authorship contribution statement

Giampaolo Bella: Conception and design of study, Acquisition of data, Analysis and/or interpretation of data, Writing – original draft, Writing – review & editing. **Pietro Biondi:** Conception and design of study, Acquisition of data, Analysis and/or interpretation of data, Writing – original draft, Writing – review & editing. **Gianpiero Costantino:** Conception and design of study, Acquisition of data, Analysis and/or interpretation of data, Writing – original draft, Writing – review & editing. **Ilaria Matteucci:** Conception and design of study, Acquisition of data, Analysis and/or interpretation of data, Writing – original draft, Writing – review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

This research gratefully acknowledges support from “Fondi di Ateneo 2020–2022, Università di Catania, linea Open Access". All authors approved the final version of the manuscript.

Appendix. Freshness value based on multiple counter

The generation and validation algorithms of FV based on Multiple Counter take into account four counters: *Trip Counter*, *Reset Counter*, *Message Counter* and *Reset Flag* (Fig. 8). These counters are increased according to specific conditions provided by AUTOSAR [16]. In particular, the Trip Counter is increased by one when the FVM starts, resets and when the power status changes. The Reset Counter is increased by one at regular time intervals. The Message Counter is increased by one value for each message transmission. Instead, the Reset Flag is represented by the two least significant bits of the Reset Counter.

It is important to note that the all counters can assume the states of *“Latest”* and *“Previous”*, more specifically:

- Latest Trip Counter or Reset Counter refer to the values received from the FVM.
- Previous Trip Counter, Reset Counter, Message Counter and Reset Flag refer to the individual freshness values used for previous authentication generation or verification.

The above counters are stored in volatile memory with the exception of the Trip Counter which is stored in non-volatile memory to reduce data loss in the event of a sudden stop of the ECU. The FVM maintains only the Trip Counter and the Reset Counter that are initialised to 1. Instead, the ECU slaves use all four counters which are initialised to 0.

A.1. Generation phase

The FV generation algorithm performs a comparison between the latest and previously sent value of both Trip Counter and Reset Counter. In particular, it checks whether the latest value is equal to the previously sent value of the two counters. Based on this comparison the FV is built. For simplicity, Table 14 shows the construction of the freshness value for the transmission. For example, if the comparison returns a positive result, then the FV is composed of the previously sent value as regards the Trip Counter and the Reset Counter, while the Message Counter will be equal to previously sent value plus one and the Reset Flag will be equal to the value from the lower end of the reset counter of the previously sent value as shown in Table 14.

A.2. Verification phase

At verification, the FVV is built and used to validate the frame. The FVV is generated by performing three comparisons: Reset Flag, Trip Counter and Reset Counter and finally on the Message Counter. Based on the result of these comparisons, the value of the receiver’s FV is constructed. For clarity, we report Table 15 (see [16]), which describes the algorighm on the possible scenarios. For example, if we consider the “Format 1” (first row of Table 15), we note that the algorithm checks whether the latest value of the Reset Flag is equal to the received value. Then, it checks whether the latest value of the Trip Counter and Reset Counter is equal to the previously received value. Finally, the previously received value of Message Counter is checked to be less that the received value. If all the checks explained above are successful, then the FVV will be composed of the previously received value as regards the Trip Counter, the Reset Counter and the Message Counter (Upper), while the Message Counter (Lower) will be equal to the received value — Message Counter Upper refers to the range that is not included in the truncated freshness value for Message Counter transmission, Message Counter Lower refers to the range that is included in the truncated freshness value for Message Counter transmission.

Table 15
Construction of Freshness Value for Reception [16].

Construction format	Condition			Construction of freshness value for verification			
	(1) Reset Flag comparison	(2) Trip Counter and Reset Counter comparison	(3) Message Counter (lower end) comparison	Trip Counter	Reset Counter	Message Counter (Upper)	Message Counter (Lower)
Format 1	Latest value = Received value	Latest value = Previously received value	Previously received value < Received value (no carry)	Previously Received value	Previously Received value	Previously Received value	Received value
Format 2			Previously received value ≥ Received value (with carry)	Previously Received value	Previously Received value	Previously received value+1	Received value
Format 3		Latest value > Previously received value	–	Latest value	Latest value	0	Received value
Format 1	Latest value-1 = Received value	Latest value-1 = Previously received value	Previously received value < Received value (no carry)	Previously Received value	Previously Received value	Previously Received value	Received value
Format 2			Previously received value ≥ Received value (with carry)	Previously Received value	Previously Received value	Previously received value+1	Received value
Format 3		Latest value-1 > Previously received value	–	Latest value	Latest value-1	0	Received value
Format 1	Latest value+1 = Received value	Latest value+1 = Previously received value	Previously received value < Received value (no carry)	Previously Received value	Previously Received value	Previously Received value	Received value
Format 2			Previously received value ≥ Received value (with carry)	Previously Received value	Previously Received value	Previously received value+1	Received value
Format 3		Latest value+1 > Previously received value	–	Latest value	Latest value+1	0	Received value
Format 1	Latest value-2 = Received value	Latest value-2 = Previously received value	Previously received value < Received value (no carry)	Previously Received value	Previously Received value	Previously Received value	Received value
Format 2			Previously received value ≥ Received value (with carry)	Previously Received value	Previously Received value	Previously received value+1	Received value
Format 3		Latest value-2 > Previously received value	–	Latest value	Latest value-2	0	Received value
Format 1	Latest value+2 = Received value	Latest value+2 = Previously received value	Previously received value < Received value (no carry)	Previously Received value	Previously Received value	Previously Received value	Received value
Format 2			Previously received value ≥ Received value (with carry)	Previously Received value	Previously Received value	Previously Received value+1	Received value
Format 3		Latest value+2 > Previously received value	–	Latest value	Latest value+2	0	Received value

References

[1] C. Valasek, C. Miller, Adventures in automotive networks and control units, 2020, URL: https://ioactive.com/pdfs/IOActive_Adventures_in_Automotive_Networks_and_Control_Units.pdf.

[2] N.I. Corporation, FlexRay automotive communication bus overview, 2020, URL: https://www.ni.com/it-it/innovations/white-papers/06/flexray-automotive-communication-bus-overview.html.

[3] Vector, Solutions for automotive ethernet, 2020, https://www.vector.com/int/en/know-how/technologies/networks/automotive-ethernet/.

[4] M. Enev, A. Takakuwa, K. Koscher, T. Kohno, Automobile driver fingerprinting, 2016, https://petsymposium.org/2016/files/papers/Automobile_Driver_Fingerprinting.pdf.

[5] M.L. Bernardi, M. Cimitile, F. Martinelli, F. Mercaldo, Driver and path detection through time-series classification, J. Adv. Transp. 2018 (2018) 1–20, http://dx.doi.org/10.1155/2018/1758731, URL: https://www.hindawi.com/journals/jat/2018/1758731/.

[6] European Union, Guidelines 1/2020 on processing personal data in the context of connected vehicles and mobility related applications, 2020, https://edpb.europa.eu/sites/edpb/files/consultation/edpb_guidelines_202001_connectedvehicles.pdf.

[7] C. Valasek, C. Miller, Remote exploitation of an unaltered passenger vehicle, 2015, http://illmatics.com/Remote%20Car%20Hacking.pdf.

[8] J. Crume, OwnStar: Yet another car hack, 2015, https://insideinternetsecurity.wordpress.com/2015/08/05/ownstar-yet-another-car-hack/.

[9] L. Constantin, Researchers hack Tesla Model S with remote attack, 2016, https://www.pcworld.com/article/3121999/researchers-demonstrate-remote-attack-against-tesla-model-s.html.

[10] Tencent Keen Security Lab, New vehicle security research by KeenLab: Experimental security assessment of BMW cars, 2018, https://keenlab.tencent.com/en/2018/05/22/New-CarHacking-Research-by-KeenLab-Experimental-Security-Assessment-of-BMW-Cars/.

[11] Tencent Keen Security Lab, Experimental security assessment on lexus cars, 2020, https://keenlab.tencent.com/en/2020/03/30/Tencent-Keen-Security-Lab-Experimental-Security-Assessment-on-Lexus-Cars/.

[12] G. Costantino, I. Matteucci, CANDY CREAM - Hacking infotainment android systems to command instrument cluster via can data frame, in: M. Qiu (Ed.), 2019 IEEE International Conference on Computational Science and Engineering, CSE 2019, and IEEE International Conference on Embedded and Ubiquitous Computing, EUC 2019, New York, NY, USA, August 1-3, 2019, IEEE, 2019, pp. 476–481, http://dx.doi.org/10.1109/CSE/EUC.2019.00094.

[13] European Union, General data protection regulation(EU regulation 2016/679), 2016, https://eur-lex.europa.eu/legal-content/EN/TXT/PDF/?uri=OJ:L:2016:119:FULL.

[14] European Union Law, UN Regulation No 155 – Uniform provisions concerning the approval of vehicles with regards to cybersecurity and cybersecurity management system, 2021, https://eur-lex.europa.eu/legal-content/EN/TXT/PDF/?uri=CELEX:42021X0387&from=EN.

[15] AUTOSAR, Requirements on secure onboard communication, 2019, URL: https://www.autosar.org/fileadmin/user_upload/standards/classic/19-11/AUTOSAR_SRS_SecureOnboardCommunication.pdf.

[16] AUTOSAR, Specification of secure onboard communication AUTOSAR CP R19-11, 2019, URL: https://www.autosar.org/fileadmin/user_upload/standards/classic/19-11/AUTOSAR_SWS_SecureOnboardCommunication.pdf.

[17] G. Bella, P. Biondi, G. Costantino, I. Matteucci, CINNAMON: A module for AUTOSAR secure onboard communication, in: 2020 16th European Dependable Computing Conference, EDCC, 2020, pp. 103–110, http://dx.doi.org/10.1109/EDCC51268.2020.00026.

[18] AUTOSAR, Specification of key manager, 2019, URL: https://www.autosar.org/fileadmin/user_upload/standards/classic/19-11/AUTOSAR_SWS_KeyManager.pdf.

[19] N. Mouha, B. Mennink, A. Van Herrewege, D. Watanabe, B. Preneel, I. Verbauwhede, Chaskey: An efficient MAC algorithm for 32-bit microcontrollers, in: A. Joux, A. Youssef (Eds.), Selected Areas in Cryptography – SAC 2014, Springer International Publishing, Cham, 2014, pp. 306–323.

[20] R. Beaulieu, D. Shors, J. Smith, S. Treatman-Clark, B. Weeks, L. Wingers, The SIMON and SPECK Families of Lightweight Block Ciphers, Report 2013/404, Cryptology ePrint Archive, 2013, https://eprint.iacr.org/2013/404.

[21] A.D. Dwivedi, P. Morawiecki, G. Srivastava, Differential cryptanalysis of round-reduced SPECK suitable for internet of things devices, IEEE Access 7 (2019) 16476–16486, http://dx.doi.org/10.1109/ACCESS.2019.2894337.

[22] IXXAT, IXXAT company, 2020, https://www.ixxat.com/products/products-industrial/tools-overview/cananalyser.

[23] SOWHAT, CINNAMON source code, 2020, https://afs.tools.iit.cnr.it/f/1c3a7f8799da4abba02c/?dl=1.

[24] S. Vaudenay, Security flaws induced by CBC padding — Applications to SSL, IPSEC, wtls..., in: L.R. Knudsen (Ed.), Advances in Cryptology — EUROCRYPT 2002, Springer Berlin Heidelberg, Berlin, Heidelberg, 2002, pp. 534–545.

[25] L. Dariz, M. Selvatici, M. Ruggeri, G. Costantino, F. Martinelli, Trade-off analysis of safety and security in CAN bus communication, in: 5th IEEE International Conference on Models and Technologies for Intelligent Transportation Systems, MT-ITS 2017, Naples, Italy, June 26-28, 2017, IEEE, 2017, pp. 226–231, http://dx.doi.org/10.1109/MTITS.2017.8005670.

[26] G. Bella, P. Biondi, G. Costantino, I. Matteucci, TOUCAN: A protocol to secure controller area network, in: Proceedings of the ACM Workshop on Automotive Cybersecurity, AutoSec@CODASPY 2019, Richardson, TX, USA, March 27, 2019, 2019, pp. 3–8, http://dx.doi.org/10.1145/3309171.3309175.

[27] A. Van Herrewege, D. Singelee, I. Verbauwhede, CANAuth-a simple, backward compatible broadcast authentication protocol for CAN bus, in: ECRYPT Workshop on Lightweight Cryptography. Vol. 2011, 2011, pp. 1–7.

[28] O. Hartkopp, C. Reuber, R. Schilling, MaCAN message authenticated CAN, 2012.

[29] A. Hazem, H. Fahmy, Lcap-a lightweight can authentication protocol for securing in-vehicle networks, in: 10th Escar Embedded Security in Cars Conference, Berlin, Germany, Vol. 6, 2012.

[30] B. Groza, S. Murvay, A. Van Herrewege, I. Verbauwhede, Libra-can: a lightweight broadcast authentication protocol for controller area networks, in: International Conference on Cryptology and Network Security, Springer, Cham, 2012, pp. 185–200.

[31] X. Ying, G. Bernieri, M. Conti, R. Poovendran, TACAN: Transmitter authentication through covert channels in controller area networks, 2019, CoRR abs/1903.05231. URL: http://arxiv.org/abs/1903.05231, arXiv:1903.05231.

[32] R. Kurachi, Y. Matsubara, H. Takada, N. Adachi, Y. Miyashita, S. Horihata, CaCAN-centralized authentication system in CAN (controller area network), in: 14th Int. Conf. on Embedded Security in Cars (ESCAR 2014), 2014.

[33] A.-I. Radu, F.D. Garcia, LeiA: A lightweight authentication protocol for CAN, in: I. Askoxylakis, S. Ioannidis, S. Katsikas, C. Meadows (Eds.), Computer Security – ESORICS 2016, Springer International Publishing, Cham, 2016, pp. 283–300.

[34] E.S. Academy, CANcrypt, 2018, https://www.cancrypt.eu/.

[35] International Organization for Standardization, Road vehicles — Controller area network (CAN) — Part 1: Data link layer and physical signalling, 2015, https://www.iso.org/standard/63648.html.

[36] T. Ziermann, S. Wildermann, J. Teich, CAN+: A new backward-compatible Controller Area Network (CAN) protocol with up to 16x higher data rates, in: Proceedings of the Conference on Design, Automation and Test in Europe, European Design and Automation Association, 2009, pp. 1088–1093.

[37] A. Bruni, M. Sojka, F. Nielson, H. Riis Nielson, Formal security analysis of the MaCAN protocol, in: E. Albert, E. Sekerinski (Eds.), Integrated Formal Methods, Springer International Publishing, Cham, 2014, pp. 241–255.

[38] International Organization for Standardization, ISO/IEC 11889-1:2015 - Trusted Platform Module library, 2015, https://www.iso.org/standard/66510.html.

[39] A. Yushev, M. Barghash, M.P. Nguyen, A. Walz, A. Sikora, TLS-over-CAN: An experimental study of internet-grade end-to-end communication security for CAN networks, IFAC-PapersOnLine 51 (6) (2018) 96–101, http://dx.doi.org/10.1016/j.ifacol.2018.07.136, URL: http://www.sciencedirect.com/science/article/pii/S2405896318308802. 15th IFAC Conference on Programmable Devices and Embedded Systems PDeS 2018.

[40] L. Dariz, M. Selvatici, M. Ruggeri, G. Costantino, F. Martinelli, Trade-off analysis of safety and security in CAN bus communication, in: The 5th IEEE International Conference on Models and Technologies for Intelligent Transportation Systems (MT-ITS 2017), IEEE, Piscataway, New Jersey, USA, 2017, pp. 226–231.

[41] L. Dariz, G. Costantino, M. Ruggeri, F. Martinelli, A joint safety and security analysis of message protection for CAN bus protocol, Adv. Sci. Technol. Eng. Syst. J. 3 (1) (2018) 384–393, http://dx.doi.org/10.25046/aj030147.

[42] B. Groza, P. Murvay, Security solutions for the controller area network: Bringing authentication to in-vehicle networks, IEEE Veh. Technol. Mag. 13 (1) (2018) 40–47, http://dx.doi.org/10.1109/MVT.2017.2736344.

[43] D. Stabili, L. Ferretti, M. Marchetti, Analyses of secure automotive communication protocols and their impact on vehicles life-cycle, in: 2018 IEEE International Conference on Smart Computing, SMARTCOMP, 2018, pp. 452–457, http://dx.doi.org/10.1109/SMARTCOMP.2018.00045.

[44] C. Bernardeschi, M. Di Natale, G. Dini, D. Varano, Modeling and generation of secure component communications in AUTOSAR, in: Proceedings of the Symposium on Applied Computing, SAC '17, Association for Computing Machinery, New York, NY, USA, 2017, pp. 1473–1480, http://dx.doi.org/10.1145/3019612.3019682.

Giampaolo Bella holds a Cambridge University Computer Laboratory Ph.D. and post-docs at Technische Universität München and Cambridge University. He has been with the University of Catania since 2001, seconded through the years with De Montfort University and SAP Research France. He is currently Visiting Professor at Royal Holloway University of London. His main research area is cybersecurity, data protection and their socio-technical aspects, having conducted inter-disciplinary research with Social Scientists, Biologists and Lawyers. He is Chief Cyber Security Advisor for ICT Cyber Consulting and directs the Catania Site of CINI Cybersecurity Lab. He is a stable EU FP7/H2020 evaluator, rapporteur and reviewer.

Pietro Biondi is a Ph.D. student in Computer Science at the University of Catania. He obtained his Master's Degree in Computer Science (summa cum laude) in July 2019 at the University of Catania. His degree thesis, concerned the study, design and implementation of a security protocol on the CAN bus called TOUCAN. From 2018 he holds a position as a junior researcher with CNR, focusing on topics related to automotive security under the supervision of Dr. Gianpiero Costantino and Dr. Ilaria Matteucci. Pietro Biondi has produced a few scientific articles on this field.

Gianpiero Costantino is a researcher at the Italian National Research Council (CNR). Currently, he has been working for the Trustworthy and Secure Future Internet group within the Institute of Informatics and Telematics located in Pisa. From November 2007 to March 2011 he was a Ph.D. student at the University of Catania. He is co-author of about fifty scientific articles. He has more than 10 years' experience in cybersecurity research and, in the last five years has focused on Automotive Cyber-security. Dr. Costantino is involved in the C3ISP, SPARTA — H2020 Projects — and WEBREPUTO — National Project —, he worked for the following project: Coco Cloud, NESSOS — FP7 — HC@WORKS-2, HC@WORKS, Trust in the Cloud — EIT Digital — Securing Smart Airport — ENISA.

Ilaria Matteucci (M.Sc. 2003, Ph.D. 2008) is a researcher of the Trustworthy and Secure Future Internet group within the Institute of Informatics and Telematics of CNR. Her main research interests include formal methods for the synthesis of secure systems, analysis of data sharing in service-oriented architectures and policies on personal data privacy. Currently, the research interest is focused on Automotive Cyber-Security, with particular reference to security properties of the CAN-bus protocol and possible vulnerabilities of Android Radio. She participates in national and European projects in the field of information security, such as FP6 EU S3MS, FP7 EU CONNECT, Consequence, Aniketos, NeSSoS, CocoCloud, Artemis EU SESAMO, H2020 C3ISP, PRIN TENACE and GAUSS, e-SHS.