Securing SOME/IP for In-Vehicle Service Protection

Marco Iorio, Massimo Reineri, Fulvio Risso, Riccardo Sisto and Fulvio Valenza

***Abstract*—Although high-speed in-vehicle networks are being increasingly adopted by the industry to support emerging use cases, previous research already demonstrated that car hacking is a real threat. This paper formalizes a novel framework proposed to provide improved security to the emerging SOME/IP middleware, without introducing at the same time limitations in the communication patterns available. Most notably, the entire traffic matrix is designed to be configured using simple high-level rules, clearly stating who can talk to whom according to the service abstraction adopted by SOME/IP. Three incremental security levels are made available, accounting for different services being associated with different requirements. The core security protocol, encompassing a session establishment phase followed by the transmission of secured SOME/IP messages, has been formally verified, to prove its correctness in terms of authentication and secrecy properties. Performance-wise, in-depth experimental evaluations conducted with an extended version of `vsomeip` confirmed the introduction of quite limited penalties compared to the bare unsecured implementation.**

***Index Terms*—SOME/IP, In-vehicle security, SOA protection.**

I. Introduction

MODERN vehicles are running on code as much as on gasoline [1], [2]. Besides the electrification trend, one of the core aspects of the 21st-century automobile relates to the development of increasingly smarter and more complex applications to manage every single aspect of the vehicle and provide a richer on-board experience. First, most operations that used to be performed through mechanical linkages are nowadays managed by electronic systems and control algorithms, according to the drive-by-wire paradigm. Second, Advanced Driving Assistance Systems (ADAS) are shifting the control of safety-critical systems, such as braking and steering, to computers, algorithms and software [3]. They aim to increase the overall safety by relieving the driver of tedious tasks, reacting faster in case of emergency and preventing accidents caused by human errors. Third, the human-vehicle interface (HMI) design is currently being reinvented, developing novel infotainment systems to provide more engaging interactions between driver, passengers and vehicle [4].

To support an increasing number of functionalities, common vehicles encompass tens of different Electronic Control Units (ECUs), ranging from low-end microcontrollers to high-performance CPU-based systems [5]. At the same time, wirings and communication protocols constitute the backbone enabling the exchange of information between distributed applications. As for in-vehicle networks, the most widespread standard is a broadcast, message-oriented bus named Communication Area Network (CAN bus) [6]. Remarkably, it adopts a lossless bitwise arbitration method of contention resolution, making it suitable for strong real-time applications.

Yet, many emerging applications, especially those from the comfort domain, are progressively associated with higher bandwidth requirements (e.g. to transmit video streams), while partially relaxing the strict real-time constraints. To this end, Automotive Ethernet [7], a slightly modified version of the widespread standard designed to meet the in-vehicle EMC requirements, is emerging prominently as a candidate to replace a plethora of complex proprietary technologies [8], [9]. At the same time, the Service Oriented Architecture (SOA) paradigm is being increasingly adopted to meet the requirements for modularity, dynamism and update capability. According to this design pattern, a system is composed of a set of services providing different functionalities, either offered or consumed by the actual applications. In this context, SOME/IP [10] is a network middleware standardized in 2016 by AUTOSAR explicitly to fulfill all the typical automotive use cases. Most notably, it provides an easy-to-use SOA abstraction on the top of traditional transport protocols, such as TCP and UDP.

Nonetheless, the SOME/IP specifications completely lack security measures directly embedded in the protocol. Although it has been designed to operate on the top of classical transport protocols, theoretically enabling the usage of effective and mature security protocol suites such as IPSec and TLS, they appear not to fit well all the different automotive communication patterns, thus introducing unwanted limitations to the middleware. IPSec, for instance, could be adopted to establish secured tunnels between different ECUs. However, being application unaware, it would not guarantee the authentication of the different parties involved in the communication. TLS, on the other hand, does not support multicast communications, leveraged by SOME/IP to limit transmissions on the communication medium when possible. Moreover, it is characterized by a fairly complex authentication handshake, being a general-purpose security protocol. Yet, during the last decade, the research community has already clearly demonstrated the need for secure in-vehicle communication protocols. Indeed, multiple researchers have discovered a wide range of possible vulnerabilities, concerning both ECUs software and in-vehicle networks (especially the CAN bus, being totally unsecured and given its relevance in the vehicular domain [11]), allowing a possible attacker to take over the control of even safety-critical systems [12]–[16]. All in all, this is a clear evidence that isolation and security through obscurity can no longer be assumed as sufficient protections.

This paper fills the identified gap by presenting a novel secu-

M. Iorio, F. Risso, R. Sisto and F. Valenza are with the Politecnico di Torino (DAUIN), Torino, Italy, {name.surname}@polito.it. M. Reineri is with Italdesign, Moncalieri, Italy, {name.surname}@italdesign.it

rity framework designed to protect SOME/IP communications. The core of our proposal, in addition to providing complete compatibility with the communication middleware, is based on two main pillars. First, the definition of simple, high-level authorization rules to specify the traffic matrix allowed in terms of services, according to the SOA pattern adopted by SOME/IP. Second, the possibility to assign different security levels to different services, depending on how critical they are. Hence, it allows to get the best compromise between security and overhead. Starting from the high-level ideas anticipated in [17], this paper presents the continuation of the above research work, which led to a rigorous definition of the framework and of its security protocol. Specifically, our main contributions are as follows: (i) The formalization of the security protocol, including the authorization mechanism, and its formal verification, to provide high assurance that the desired security properties hold. (ii) An in-depth presentation of the security framework architecture envisioned to protect SOME/IP communications, discussing the automotive requirements and justifying the possible design choices. (iii) The extensive experimental assessments of the overhead introduced by the security features, carried out using automotive-grade evaluation boards running the developed PoC.

The remainder of the paper is organized as follows. Section II reviews previous cyber-attacks and existing solutions to protect in-vehicle networks. Section III presents an overview of the system model increasingly adopted in modern vehicles, with specific focus on the SOME/IP middleware. Section IV presents the proposed security framework, together with a discussion the security properties it can enforce. Section V formalizes the security protocol designed to protect SOME/IP communications, which is formally verified in Section VI. Section VII outlines the advantages of our proposal compared to the usage of SOME/IP over a lower secure protocol, while Section VIII presents its experimental evaluation. Finally, Section IX draws conclusions and proposes directions for further research.

II. Related Work

In this section, we motivate the requirements for in-vehicle security, presenting an overview of the main attacks perpetrated against vehicular networks. Then, we proceed with an overview of the possible solutions proposed by the research community.

A. In-Vehicle Networks Attacks

During the last decade, multiple researchers have already analyzed the security of in-vehicle networks. Yet, their conclusions are discouraging, pointing out the existence of multiple vulnerabilities that could eventually enable a sufficiently skilled attacker to take over the control of even safety critical systems. In 2008, Hoppe et al. [12] depicted four different attack scenarios that, exploiting messages injected into the CAN network, allowed the researchers to perform simple actions, such as opening the window lift or hiding an incomplete repair. Two years later, Koscher et al. [13] moved on and showed they could control the display of the speedometer, kill the engine, as well as affect braking by simply injecting messages into the CAN bus of a vehicle. The research received widespread criticism because people claimed there were no ways for an attacker to inject these types of messages without being close to the vehicle. Nonetheless, the same research group [14] succeeded in remotely performing similar attacks, by exploiting interfaces such as the MP3 parser of the radio, the Bluetooth stack and the telematics unit to get the code executed. Finally, in 2015, Miller and Valasek clearly demonstrated that remote car hacking of an unaltered vehicle was indeed possible [15]. In a nutshell, they leveraged a chain of serious vulnerabilities in applications and network stack implementations, especially those from the comfort domain (e.g. Bluetooth, Wi-Fi and 4G) to remotely hack the infotainment system of a 2014 Jeep Cherokee. Eventually, they succeeded in remotely reprogramming the firmware of a microcontroller to forward messages to the CAN bus, gaining the capability to perform a wide range of physical actions through a laptop, wherever in the US. In 2016, adopting a similar approach, Nie et al. [16] successfully implemented a remote attack on a Tesla Model S in both Parking and Driving mode.

B. In-Vehicle Networks Security

The security of in-vehicle networks has already been the topic of many studies. Being almost ubiquitous, the CAN bus has received widespread attention to increase the overall security and prevent malicious intrusions [18]–[27]. Using cryptography at the application layer is the most natural choice when it comes to protect network messages. However, due to the broadcast nature of the CAN bus, as well as the computational constraints imposed by low-end ECUs, most current approaches propose the usage of simple Message Authentication Codes (MACs) to authenticate the messages transmitted [20], [26].

In this context, the biggest challenge resides in the key provisioning phase. Although public key cryptography is usually considered out of scope due to the excessive requirements in terms of computational capacity [25], fixed and statically assigned keys are not valuable either, being very easy to compromise. A possible solution to the key exchange problem involves short-term keys released in a time dependent fashion, as done by the Timed Efficient Stream Loss-Tolerant Authentication (TESLA) protocol [28]. Nonetheless, this approach introduces a small delay between the reception of a message and the actual verification instant, as well as requires the messages to be buffered. Conversely, other researchers proposed to move from pair-wise keys to a group-based approach. In other words, acknowledging the one-to-many nature of the communication medium, they suggested the usage of group keys, depending on the trust level associated with each ECU [20], as well as the exploitation of an advanced MAC construction to mix the keys between groups of nodes [18]. Finally, another subset of solutions leverages the physical properties of the signals transmitted on the CAN bus and the ECUs characteristics as unique patterns in the generation and exchange of the symmetric keys [24], [27].

To remove the need for cryptography, along with its associated complexity and overhead, some researchers started exploiting the unique characteristics of CAN signals as an authentication mechanism. In particular, they showed it was

possible to obtain a fingerprint of the sender node by applying mathematical functions on the electrical signal characteristics [19], as well as by measuring the clock drifts in the communication [23]. Although based on empirical measurements, these approaches may provide a valuable solution to detect undesired modifications to the CAN bus topology, including the addition and the replacement of an ECU.

Acknowledging the typical usage of a centralized gateway to interconnect the different network technologies coexisting in a single vehicle, Wolf et al. [29] proposed in 2006 to leverage the gateway itself as a firewall to validate the messages exchanged. In particular, they argued the necessity to enforce firewall rules to prevent ECUs attached to lower restricted networks from sending messages into safety-relevant bus systems, such as CAN. Nonetheless, different attacks showed this protection alone not being sufficient, given the high attack surface exposed by the gateway [15], [16]. Indeed, the central gateway usually acts also as a terminator for the connections from and to the external world (e.g. Bluetooth, Wi-Fi and 4G), thus becoming a very easy target for an attacker. Additionally, although a firewall could provide isolation between different domains, each sub-network would still remain totally unprotected.

When it comes to in-vehicle Ethernet-based communications, Hamad et al. [30] proposed in 2017 a framework that aims to provide secure communications between ECUs by exploiting security policies to define who should talk to whom. Their solution is made up of two main building blocks. First, a framework used to build communication policies gradually throughout the design and lifecycle of the software component, modelling trust relationships through a Public Key Infrastructure (PKI). Second, a security module enforcing the policies in a distributed manner. Nonetheless, it requires the definition of low-level rules, strongly limiting the dynamism introduced by SOAs. Additionally, the verification of lengthy chains of trust may impose an excessive burden at start-up time.

The same year, Zelle et al. [31] studied the feasibility of using TLS to guarantee authentication and confidentiality to Ethernet-based in-vehicle communications. In the end, they concluded that TLS could meet most of the real-world performance requirements using typical automotive hardware, especially when evaluating the actual run-time protection. A single session establishment, however, took more than $2\,\mathrm{s}$ to complete, rising many doubts whether it could be sustainable in case hundreds of connections needed to be established in parallel. Additionally, they only considered the exploitation of TLS to protect TCP communications, while leaving the use of UDP and DTLS as a future work. Indeed, UDP appears to be the privileged transport protocol to be used in conjunction with SOME/IP, being particularly lightweight. Nonetheless, as of today, no standard version of DTLS does provide compatibility with multicast communications, usually leveraged in vehicular networks.

III. Background

This section introduces the basic vehicular system architecture, focusing on high-performance ECUs, and presents the SOME/IP middleware, its open-source implementation `vsomeip`, as well as an alternative solution, DDS.

A. System Model

Typical vehicles consist of roughly 100 ECUs, microcontrollers responsible for specific tasks, such as engine control, power train, body control and so on. Each functional ECU is directly attached to multiple sensors and actuators, while their interconnection is realized through multiple CAN buses.

Yet, emerging use cases, including automated driving, high-speed on-board and vehicle-to-vehicle (V2V) communication, multimedia applications and continuous over-the-air updates, are pushing towards the so-called "central server" architecture [32]. According to this paradigm, many different ECUs are replaced with few more complex devices, usually consisting of a high-performance microprocessor complemented by multiple microcontrollers, interconnected through high-speed Ethernet links. Sensors and actuators can be attached directly to the network or through integration nodes, becoming more and more ECU and vehicle independent.

To increase the interoperability between multiple devices and allow for a faster development of the applications, the AUTOSAR consortium recently standardized Adaptive Platform [33], which is becoming the de facto standard for high-performance ECUs. It seeks to provide a common middleware interface between applications and ECUs on top of POSIX-compliant operating systems (e.g. Linux, PikeOS or QNX), to enable the efficient usage of the underlying hardware resources and abstract the low-level details. According to the traditional approach (i.e. Classical AUTOSAR), each application is statically assigned to a single ECU and the traffic matrix is defined at the time of configuration. Conversely, Adaptive AUTOSAR adopts a more flexible solution, leveraging the service-oriented pattern to decouple the logical high-level goals from the actual implementations and the network topology.

B. SOME/IP

Scalable service-Oriented MiddlewarE over IP (SOME/IP) is a communication middleware standardized by the AUTOSAR consortium as part of its effort to develop a future-proof on-board architecture [10]. To achieve isolation and modularity, SOME/IP adopts a service-oriented abstraction. A service logically represents a business activity with a specified outcome, accessible through a well-defined interface. In other words, consumers perceive each service as a "black box". Services can be built upon other services, as well as combined to provide more complex functionality. Additionally, different instances of the same service may coexist at the same time (e.g. for redundancy purposes), and reside on different ECUs as well as on the same device. Finally, service discovery functionalities are foreseen to advertise the availability of the different services and their associated network parameters [34].

SOME/IP provides two main communication models. First, request/response, implementing classical Remote Procedure Calls (RPC) to invoke functions exposed by applications running on remote devices. Second, the publish/subscribe approach, which is rather typical in automotive networks. It decouples the sender from the recipients of the messages: whenever an event occurs, the corresponding service publishes a new notification; applications willing to receive updates, on

the other hand, express their interest by subscribing to the event. The actual messages are delivered seamlessly by the middleware, which can leverage all the features offered by lower network layers (e.g. multicast messages) to save transmissions on the communication medium.

SOME/IP operates on the top of a transport protocol, in charge of delivering the messages from the sender to the recipient(s). Two main bindings are currently supported by the specifications: UDP and TCP. Generally speaking, UDP shall be the preferred binding, introducing few overhead and being suitable also in case of hard latency requirements. Additionally, it supports multicast messages, thus optimizing the network utilization. On the other hand, being a heavyweight protocol, TCP is suggested only in case very large chunks of data need to be transported and no hard latency requirements in case of error exist. Other transport mechanisms, such as Network File System (NFS) or Automotive Pixel Link, could be used if more suited for the specific use case. As for message protection, the SOME/IP standard does not integrate any security measures, including authentication, integrity and confidentiality. Instead, they are delegated to the transport layer, which may impose unwanted limitations and introduce excessive overhead.

C. vsomeip

The `vsomeip` stack[1] is an open-source implementation of the SOME/IP specifications. In a nutshell, the library provides all the functionalities required to abstract the communication between both local and remote applications, including serialization of message headers, establishment of connections, management of communication endpoints, packets transmission and service-discovery facilities. In addition, it exposes a public API to the application developers, taking advantage of an event-loop abstraction, as well as leverages configuration files to allow fine-tuning a wide range of parameters.

The core of `vsomeip` resides in the concept of routing manager, i.e. the entity responsible for the effective delivery of network messages. Two different versions of the routing manager entity do coexist and are loaded transparently by the library at start-up time:

- *routing manager*, the fully functional version of the module, adopted by a single application for each physical ECU. It acts as a gateway in charge of the communication with applications residing on remote devices. In other words, this is the only instance establishing external network connections and managing the transport endpoints (i.e. UDP and TCP sockets). Finally, it takes care of loading the service discovery module and internally propagating the information received.
- *routing manager proxy*, loaded by all the other instances and capable of local communications only (i.e. implemented through Unix domain sockets). Hence, messages addressed to remote recipients are first transmitted to the main routing manager, which forwards them to the intended destination. There, the inverse process may need to be executed. Service messages are continuously exchanged between the proxies and the master instance, to share the knowledge on offered and requested services and to allow the delivery of notifications.

As for security, `vsomeip` can optionally leverage Unix credentials to authenticate local connections. However, the total lack of protection regarding network messages, as well as the usage of unauthenticated configuration files, restrict the usability of this solution to a rather limited niche of use cases.

D. DDS

Data Distribution Service (DDS) [35] is a data-centric communication middleware based on the publish/subscribe pattern and standardized by the Object Management Group (OMG) in 2004 (the latest version, 1.4, was released in 2015). It targets distributed real-time systems in the industrial IoT domain and it is currently adopted in many verticals, including transportation, energy, medical systems, industrial automation, aerospace and defence.[2] At its core, DDS is based on topics, an abstraction associating a unique name and a data type to the actual data: writers publish content in a topic that readers can subscribe to, getting notified of its availability. Each topic can be assigned a set of Quality of Service (QoS) descriptors, to characterize reliability, security and storage requirements. Additionally, dynamic discovery functionalities are featured to enable "plug-and-play" DDS applications. Finally, although strongly oriented towards publish/subscribe communication, in 2017 the DDS specifications were extended to implement classical RPC on top of its basic building blocks [36].

To exchange the actual information on the network, DDS leverages a custom wire protocol named Real-Time Publish Subscribe (RTPS) [37]. It introduces an abstraction layer implementing transport-agnostic reliability and fragmentation features on top of classical protocols, such as TCP and UDP, hence providing reliable channels even on top of connectionless and multicast IP communications. The DDS approach to security is also transport-agnostic, being based on an extension of the RTPS capabilities [38]. In a nutshell, it aims to enforce authentication and authorization of DDS readers and writers, as well as integrity and confidentiality of the data exchanged.

DDS appears to a be viable alternative for the automotive domain, and it is also characterized by some benefits over SOME/IP, QoS management and security among all. To this end, although the AUTOSAR `ara::com` communication management API designed as part of the Adaptive Platform was initially modeled around SOME/IP concepts (i.e. methods, events and fields, as well as its service discovery protocol), the support for DDS has been introduced in newer versions. Yet, SOME/IP still appears to hold a privileged position in the automotive domain, thanks to its support for simpler ECUs running Classic AUTOSAR, as well as its tighter integration within the AUTOSAR framework. For these reasons, we believe an improvement in the SOME/IP security to be of paramount importance and in the remainder of the discussion we will focus explicitly on this middleware, while leaving a complete comparison between the two solutions as a future work.

[1] https://github.com/GENIVI/vsomeip

[2] https://www.dds-foundation.org/who-is-using-dds-2

IV. Securing SOME/IP

This section presents the key characteristics of our security framework, which mainly consists of a two-phase security protocol meant to provide improved security to SOME/IP. It is composed of an initial session establishment phase, performed at start-up time between each application interested in accessing a SOME/IP service and the corresponding offerer. Asymmetric cryptography is leveraged to ensure that only authorized parties can start the communication, as well as to exchange the data necessary for the subsequent protection. Once the session establishment correctly completed, the actual run-time protection can take place. Symmetric cryptography is adopted to enforce the efficient exchange of secured SOME/IP messages, according to the security level selected in the previous phase.

Our solution was driven by the desire to achieve complete compatibility with the communication middleware, avoiding to introduce any limitations that may constraint the dynamism typical of SOME/IP. First, security features shall be as much transparent as possible from the applications point of view. In other words, the definition of the permitted traffic matrix shall be performed by means of high-level authorization rules (i.e. in terms of services), without having to deepen the network parameters abstracted by the middleware. Second, the protection shall be compatible with all the communication models supported by SOME/IP, regardless of the transport protocol adopted for the actual delivery of the messages. In particular, it shall transparently cope with both one-to-one (i.e. unicast) and one-to-many (i.e. multicast) transmissions, the latter being exploited for the efficient delivery of notifications.

The remainder of this section is organized as follows. We start delineating the adversary model assumed for the rest of the discussion. Then, we outline the most noteworthy characteristics of the security framework, focusing on the granularity adopted and motivating the available security levels. Additionally, we discuss the methodology proposed to express at a high level the authorization rules, as well as the strategy selected to secure multicast messages. Finally, Section V formally describes the actual security protocol proposed to secure SOME/IP messages.

A. Threat model

In this work, we assume the adversary may have full control over the Ethernet network, being able to insert, replay, eavesdrop, modify and drop arbitrary messages. Yet, we do not consider Denial-of-Service (DoS) attacks, leaving their protection as future work. As for cryptography, we assume the adversary has bounded computational power. Specifically, he is assumed to be unable to subvert widely adopted cryptographic functions or break sufficiently long secrets by brute force.

For the purpose of our analysis, it is not relevant how the attacker obtained its access, e.g. through software compromise or by connecting a physical device to the in-vehicle network, for tuning reasons or by a dishonest repair shop. Nonetheless, we recognize that ECUs are completely accessible by possible hackers, who can study and practice multiple attacks on their own vehicles before moving to the actual targets. Hence, to achieve a sufficient level of protection, we suppose each ECU is equipped with a hardware secure storage, preventing an attacker to access the cryptographic material. At minimum, the secure storage shall securely store and bind each private key to a specific application, as well as guarantee the integrity of critical files (e.g. the root digital certificate). Yet, the usage of hardware security modules in the automotive industry has already been envisioned quite a long time ago [39].

B. Security Granularity

Compatibly with the service-oriented paradigm adopted by SOME/IP, our security framework operates at service instance granularity. In other words, every instance of a SOME/IP service is an atomic entity which, from the security point of view, every application is either allowed or denied access to. This granularity level originates as a trade-off between strong isolation, pushing towards a very fine discrimination, and the constrained resources available in vehicular networks. Yet, it is deemed not to introduce any particular limitations. First, services represent just a logical abstraction on the top of the applications implementing the actual functionalities, i.e., multiple services can be provided by the same application. Hence, the security granularity ultimately depends on the architectural choices made by the designers in charge of deciding what to call a service. Second, services should inherently group together closely related operations, intuitively associated with also similar security constraints and accessible by the same requesters. Yet, it might be reasonable to divide the functions only returning pieces of information (i.e. getters and events), from those modifying the environment, the latter being possibly restricted to a narrower group of users.

C. Security Levels

Different services may be associated with different requirements in terms of security, depending on the functionalities they provide and the corresponding degree of criticality. Accounting for multiple use cases, our framework provides three incremental security levels, namely *nosec*, *authentication* and *confidentiality*. They can be independently assigned to each instance of a service, to achieve the best trade off between protection and overhead.

1) Nosec: it is the simplest security level, merely corresponding to vanilla SOME/IP. Albeit not providing any security property, it may be suitable for very simple use cases, since it introduces no additional communication overhead. Additionally, it ensures full compatibility with legacy applications, imposing no modifications to the message format.

2) Authentication: it provides data authentication and integrity. Specifically, it ensures that the middleware processes only messages originating from a legitimate source (i.e. authentication and authorization), given they have not been modified while flowing through the network (i.e. integrity). Additionally, this security level provides replay protection. In other words, it identifies and drops duplicate packets, preventing an attacker from capturing valid messages to subsequently retransmit them and trigger multiple times the same action, as well as cause application crashes. Intuitively, the security properties enforced by the *authentication* level are of the highest importance whenever a communication takes place. Yet, they

are even more imperative in a vehicular network, being cars safety-critical systems. The higher the level of automation, the more serious the potential effects: in the automated driving era, companies might be considered liable in case of incidents caused by malfunctioning software.

3) Confidentiality: it ensures authentication, authorization, integrity and confidentiality, as well as replay protection. In other words, it includes all the security properties introduced by the *authentication* level, complemented by encryption to prevent a possible attacker from accessing the contents of the messages transmitted across the network (i.e. providing data confidentiality). Different use cases may justify the introduction of data confidentiality. First, in-vehicle networks are no longer confined to sensor data. Indeed GPS positioning, advanced infotainment systems, as well as the interconnection with personal devices (e.g. smartphones), produce a great amount of valuable data, possibly revealing privacy-sensitive information about the driver and the passengers. For instance, a very simple device may be able to track all the movements of a vehicle, simply eavesdropping the periodic messages advertised by the navigation system. Additionally, car manufacturers may be interested in ensuring data confidentiality to prevent industrial espionage. Indeed network messages may provide important hints to reconstruct the logic of complex and possibly distinctive systems, especially those related to perception and automated driving. Finally, encryption might be leveraged to forbid non-authorized aftermarket ECUs from reading the messages exchanged between the other devices. Hence, it provides a way to selectively limit their ability to grab external information.

D. Expressing the Traffic Matrix

One of the most distinctive aspects of our security framework resides in the usage of simple high-level authorization policies to express the permitted traffic matrix, i.e. the set of connections that are allowed to be established by the middleware. Sticking to the SOME/IP jargon, we delineate the conceptual format of each independent authorization rule as the 5-tuple

$$\{app, srv, inst, role, min_{SL}\} \tag{1}$$

where:

- app identifies the application to which the rule applies;
- srv points out the SOME/IP service the rule refers to;
- $inst$ specifies either an explicit instance of srv to which the rule applies, or contains a wildcard to extend its validity to any instance;
- $role$ defines the role app can assume for the considered service instance, among *offer* (i.e. implement the interface defined by the service) and *request* (i.e. consume the functionalities exposed by the service); the offerer of a service instance is automatically allowed to request it;
- min_{SL} states the minimum security level, among *nosec*, *authentication* and *confidentiality*, the application requires to be enforced when accessing the service instance.

In other words, each policy states that an application app is authorized to access a specific service instance $\{srv, inst\}$ with a given $role$ (i.e. offer or request), granted that its security level is at least min_{SL}. An unbounded number of rules can be combined, ending up defining the entire set of communications that can occur between the applications hosted in a vehicle (i.e. a white-listing approach is adopted).

Being the session establishment phase carried out by means of asymmetric cryptography, each application is required to be associated with a private key, sealed within the secure storage, and a digital certificate, embedding the corresponding public key. Digital certificates need to be signed to guarantee the authenticity and integrity of their content, according to a chain of certification. Without loss of generality, in the following we consider a simplified chain, composed by no intermediate levels between the leaves and the trusted root. Yet, to achieve the best degree of security, we suggest using a different root certificate for each vehicle. Thus, even if an attacker eventually succeeds in obtaining a valid private key (e.g. by violating the secure storage of an ECU), he can leverage it only on the original device, being the corresponding certificate recognized as invalid by all the other vehicles. Certificate revocation is currently considered a critical task in the vehicular domain, due to the requirements for an Internet access and the rapidly growing size associated with CRLs. Hence, per-vehicle certificates, required to be periodically renewed, are deemed to represent a good compromise between security and complexity.

In light of this, we propose to leverage the digital certificates themselves as trustworthy anchors to express the set of services each application is authorized to access. In particular, in the following we delineate two possible alternatives, highlighting the most relevant differences between them.

1) Centralized approach: each digital certificate binds a public key to the identifier of the application it is associated to. Then, a centralized repository available on each vehicle is leveraged to store the entire set of relevant authorization rules according to the format defined by (1), where app corresponds to the identifier contained in a certificate. Additionally, a digital signature enforces the authenticity and integrity of the rule set, to detect any possible tampering attempts. Whenever a new secure connection has to be established, the framework performs two lookups in the repository, to verify whether both parties are authorized to participate in the communication. According to this solution, each vehicle is characterized by a single source of trust, stating the entire traffic matrix. Still, the repository might be replicated on every ECU, to simplify its secure access and limit the communication latency. Nonetheless, whenever, e.g., a new application is installed or an update involves a modification in the rule set, the entire repository needs to be rebuilt, signed by the car manufacturer and stored on the target vehicle, possibly introducing a quite high overhead.

2) Distributed approach: each digital certificate directly binds a public key to the rule set associated with that specific application. Technically speaking, with reference to the X.509 standard, we propose to store each authorization rule as a Subject Alternative Name (SAN), adopting a URI-like format:

$$\texttt{someip} : srv : inst \ / \ role = min_{SL} \tag{2}$$

where `someip` is a keyword characterizing the type of entry and app is implied by the owner of the private key associated with the certificate. Although having no longer a central repository containing the entire set of rules, the distributed

approach is characterized by a few advantages over a centralized solution: (i) The framework does not need to perform any table lookup when establishing the connection, being all the required pieces of information contained by the certificates it is presented to. (ii) Installing or updating an application does not require to regenerate the entire rule set, but only to issue a new digital certificate containing the updated rules relevant for that application. Indeed, the authorizations associated with all the other applications still remain valid, as they are bounded to SOME/IP services and do not imply the identity of the actual counterpart. As a practical example, let's suppose an upgrade is installed in a vehicle, introducing a new rear-view camera and its companion application. This application is authorized to offer an *rwc-stream* service to transmit the video stream, as well as to send alerts to a *collision-alert* service signaling possible upcoming dangers. Assuming the central dashboard was already offering the *collision-alert* service to receive alerts from other sources, it will transparently get notifications from the new application too, being the access compatible with its digital certificate. Conversely, if no support was provided to read data from the *rwc-stream* service, the displaying application would require an update to introduce this capability, as well as to regenerate its digital certificate accordingly. Long story short, the car manufacturer is required to issue an updated digital certificate only when the corresponding application is also updated. (iii) Certificates can be leveraged by car makers as a sort of contract, to certify that a given application, optionally developed by a third-party, is authorized to request and/or offer a predetermined set of service instances. Thanks to the increased modularity and dynamism, we will stick to the distributed approach for the rest of the discussion. Nonetheless, in case deemed appropriate, the security framework could be trivially adapted to support the centralized solution.

E. Symmetric Keys Granularity

Generally speaking, SOME/IP service instances are associated with a one offerer, many requesters topology. They possibly encompass unicast interactions between each separate requester and the single offerer, as well as group notifications transmitted by the provider to (a subset of) the requesters. With that in mind, the security framework needs to be compatible with both unicast and multicast messages, in order not to impose any limitations on the SOME/IP middleware.

Run-time messages protection has been designed to be efficiently provided by means of symmetric cryptography. Hence, two possible alternatives open up regarding the granularity of the symmetric keys associated with each service instance. First, the framework may leverage a different symmetric key between the offerer and each requester, to secure the direct communications. An additional key should then be shared among all the applications accessing the service instance, to cope with multicast traffic (optionally, to increase even further the granularity, different keys might be used for different events). Second, a single symmetric key could be leveraged to secure both unicast and multicast messages associated with a specific service instance. The former alternative should provide a higher level of security, guaranteeing a one-to-one authentication of unicast messages and preventing direct interactions between different requesters. Nonetheless, a malicious requester could anyway easily compromise the entire service, directly triggering unwanted actions or transmitting counterfeit notifications. Hence, as a trade-off between security and complexity, we adopted the second approach, providing a group-level protection. Symmetric keys are automatically generated by the framework every time a new service instance is started. Thus, each key is deemed to be used for a limited time, significantly reducing the possibilities for a successful attack. Yet, a re-keying mechanism might be required in case of long-running services, according to a well-established practice.

V. Security Protocol

This section presents in great detail the security protocol at the core of our solution, characterized by an initial handshake phase for session establishment followed by the transmission of secured messages. After an initial set of definitions, we proceed with a formal description of the security protocol, laying the foundations for the subsequent formal verification.

The protocol will be described at symbolic level, i.e. abstracting away some details of the cryptographic operations used (e.g. the specific cryptographic algorithms or the key lengths). This description is enough for the formal verification of the protocol logic, according to the Dolev-Yao approach [40].

A. Preliminaries

In the following, we define the terminology adopted to formalize the security protocol herein presented. For the sake of clarity, and without loss of generality, we temporarily assume the existence of a single service instance to be secured, identified by the pair $\{srv_x, inst_x\}$. Yet, the protection can be trivially extended to the realistic case of multiple service instances coexisting in the same system. Let A be the single application authorized to offer $\{srv_x, inst_x\}$, while $\{B_i\}$, $\forall i \in [1, n]$, represents the set of requesters interested in accessing the given service instance, for some $n \geq 1$. Additionally, let us assume A being associated with a private key sk_A and a digital certificate $cert_A$, embedding both the corresponding public key pk_A and an authorization policy ξ_A specified according to (2), for a given minimum security level σ_A among *nosec*, *authentication* and *confidentiality*:

$$\xi_A = \{\texttt{someip} : srv_x : inst_x \,/\, \textit{offer} = \sigma_A\} \tag{3}$$

Similarly, let each requester B_i be linked to a private key sk_{B_i} and the related digital certificate $cert_{B_i} = \{pk_{B_i}, \xi_{B_i}\}$, with:

$$\xi_{B_i} = \{\texttt{someip} : srv_x : inst_x \,/\, \textit{request} = \sigma_{B_i}\} \tag{4}$$

Additionally, let $cert_R$ identify the root certificate. Finally, for a given service instance, a security level sl is said to be *acceptable* by an application if and only if $sl \geq \sigma$, where σ is the minimum security level stated in the authorization policy ξ. For instance, if ξ_A states $\sigma_A =$ *authentication*, then both *authentication* and *confidentiality* levels would be *acceptable* for A. Conversely, *nosec* would not.

Algorithm 1 Session establishment: Offerer A

Require: $srv_x, inst_x, sk_A, cert_A, cert_R$
1: $\xi_A \leftarrow$ get-policy$(cert_A, srv_x, inst_x)$
2: $sl \leftarrow$ security-level$(\xi_A,$ user-configuration$)$
3: $srvkey \leftarrow$ rndkey$()$
4: $peerid \leftarrow 1$
5: **loop**
6: **in**(Request, $\{srv_x^*, inst_x^*, n, cert_{B_i}\}$)
7: **if** $\{srv_x^*, inst_x^*\} \neq \{srv_x, inst_x\}$ **or**
8: $\longrightarrow$ **not** verify-cert$(cert_{B_i}, cert_R)$ **then**
9: **abort**
10: **end if**
11: $\xi_{B_i} \leftarrow$ get-policy$(cert_{B_i}, srv_x, inst_x)$
12: **if not** verify-policy$(\xi_{B_i}, request, sl)$ **then**
13: **abort**
14: **end if**
15: $pk_{B_i} \leftarrow$ get-pubkey$(cert_{B_i})$
16: $enckey \leftarrow$ aenc$(srvkey, pk_{B_i})$
17: $response \leftarrow srv_x, inst_x, n, cert_A, peerid, sl, enckey$
18: $signature \leftarrow$ sign$(response, sk_A)$
19: **out**(Response, $\{response, signature\}$)
20: $peerid \leftarrow peerid + 1$
21: **end loop**

Algorithm 2 Session establishment: Requester B_i

Require: $srv_x, inst_x, sk_{B_i}, cert_{B_i}, cert_R$
1: $n \leftarrow$ nonce$()$
2: **out** (Request, $\{srv_x, inst_x, n, cert_{B_i}\}$)
3: **in** (Response, $\{response, signature\}$)
4: $srv_x^*, inst_x^*, n^*, cert_A, peerid, sl, enckey \leftarrow response$
5: $\xi_B \leftarrow$ get-policy$(cert_B, srv_x, inst_x)$
6: **if** $\{srv_x^*, inst_x^*\} \neq \{srv_x, inst_x\}$ **or** $n^* \neq n$ **or**
7: $\longrightarrow$ **not** verify-policy$(\xi_B, request, sl)$ **or**
8: $\longrightarrow$ **not** verify-cert$(cert_A, cert_R)$ **then**
9: **abort**
10: **end if**
11: $\xi_A \leftarrow$ get-policy$(cert_A, srv_x, inst_x)$
12: **if not** verify-policy$(\xi_A, offer, sl)$ **then**
13: **abort**
14: **end if**
15: $pk_A \leftarrow$ get-pubkey$(cert_A)$
16: **if not** verify-signature$(signature, pk_A)$ **then**
17: **abort**
18: **end if**
19: $srvkey \leftarrow$ adec$(enckey, sk_{B_i})$
20: **return** $srvkey, peerid$

B. Session Establishment

The first pillar characterizing the security protocol proposed to protect SOME/IP communications consists of message handshakes to establish the secure sessions. A handshake is transparently carried out by the framework between each application willing to access a specific service instance and the corresponding offerer. Indeed, although secure sessions operate at group level to account for one-to-many communications, each handshake is a one-to-one authentication process, performed independently between the application A, offering $\{srv_x, inst_x\}$, and each requester B_i, $\forall i \in [1, n]$ (i.e. n separate handshakes are performed by the framework at start-up). The entire handshake process is repeated for each unique service instance, regardless of whether more instances are offered by the same or by different applications.

The session establishment phase serves two main purposes. The first purpose is to perform a mutual authentication, and to ensure that each application accesses only service instances complying with the authorization policies stated by its corresponding digital certificate. More precisely, the framework automatically enforces the role associated with the application (i.e. *offer* or *request*) and verifies whether the current security level of the service instance (i.e. the one decided by the offerer, as detailed in the following) is *acceptable*. In order to get the offerer's authentication, the offerer digitally signs its handshake response with its own private key, and the requester verifies this signature. The requester, on the other hand, gets authenticated implicitly, since it needs its private key to decipher and retrieve the symmetric key used for the subsequent run-time protection. The second purpose of the handshake is to share the session parameters between offerer and requester, including the symmetric key, transmitted in encrypted form.

Algorithms 1 and 2 formalize respectively the operations performed by the framework on behalf of the offerer A and a requester B_i to establish a secure session. Each algorithm assumes to receive in input the service instance for which the handshake is to be performed (i.e. $\{srv_x, inst_x\}$), the private key and the digital certificate associated with the current application, as well as the root certificate $cert_R$.

Focusing initially on the offerer, upon startup the framework selects a security level ($sl \geq \sigma_A$) to be associated with the service instance, compatibly with the corresponding authorization policy ξ_A and optionally depending on a user-defined configuration (granted that it is *acceptable*). Indeed, an application developer may desire to select a security level higher than σ_A, in order to make it *acceptable* by a wider range of requesters. Additionally, the framework generates a new random symmetric key ($srvkey$), to be adopted for the run-time protection of $\{srv_x, inst_x\}$. Indeed, sticking to group protection, most security parameters (e.g. security levels and cryptographic algorithms) need to be shared by all the applications accessing the same service instance. Hence, they are decided by the offerer and communicated to the other members during the session establishment phase.

Once initialization is terminated, the offerer is ready to accept authentication requests, as represented by the infinite loop at lines 5–21. The actual session establishment is started by the requester, which initially generates a new random nonce n leveraged to associate each request to the corresponding response and prevent message replay. Then, it builds up the authentication request message, identifying the target service instance and containing the digital certificate associated with the requester. Upon reception, the offerer performs some sanity checks, verifies the validity of the digital certificate and the fulfillment of the authorization policy ξ_{B_i}. In particular, it ensures that B_i is authorized to access $\{srv_x, inst_x\}$ as a requester, as well as that the security level sl is *acceptable* according to ξ_{B_i}. In other words, it prevents an application from accessing a service instance less secure than required. In case an inconsistency is identified during any step, the handshake is immediately aborted. If all checks pass, the framework leverages asymmetric cryptography to encrypt the symmetric key with the public key extracted from the digital certificate of the requester (lines 15–16). Hence, only the owner of the corresponding private key would be able to decipher it. The

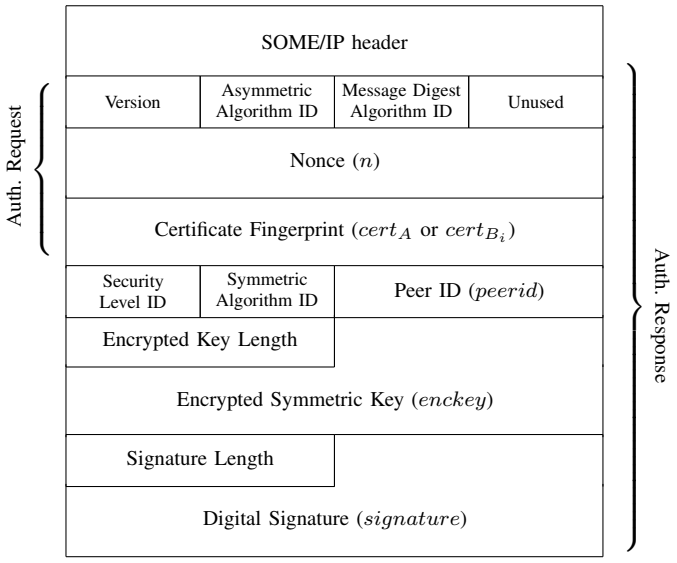

Fig. 1. On-wire format of the SOME/IP messages exchanged during the session establishment phase: the request encompasses the fields up to *Certificate Fingerprint*, while the response includes all the elements presented.

response message is then constructed, mainly including the nonce copied from the request, the certificate of the offerer, the encrypted key, the security level selected by the offerer and an identifier ($peerid$) uniquely assigned to each requester (the offerer has $peerid = 0$). Finally, the response is digitally signed with the private key sk_A of the offerer (line 18), to ensure its authenticity and integrity. In the end, the message is sent back to the requester.

Once the response is received, the requester performs again a set of checks, verifying whether the security level sl selected by the offerer is acceptable according its own authorization policy ξ_B, and ensuring the validity of $cert_A$. Additionally, it extracts the offerer's authorization policy ξ_A to make sure A has $role$ equal to *offer* and sl is *acceptable* for the offerer. Indeed, no legitimate requester would access $\{srv_x, inst_x\}$ if $sl < \sigma_A$, to prevent an offerer from violating its constraints. Then, the digital signature is verified by means of the public key extracted from $cert_A$, explicitly authenticating A (line 16). In case of successful outcome, the framework can proceed deciphering $enckey$ by means of its private key (line 19). Hence, the requester finally obtained the symmetric key leveraged to secure run-time messages. No confirmation is sent to the offerer, thus achieving an implicit authentication. Nonetheless, a malicious application would not own the correct private key sk_{B_i}, effectively being prevented from decrypting the symmetric key and participating in the subsequent communication.

The session establishment phase is transparently executed by the security framework on behalf of the actual applications, which are notified of the availability of a service instance only after successfully completing the entire process. Indeed, the SOME/IP middleware is exploited for the efficient delivery of the handshake messages, by leveraging the request/response pattern. Fig. 1 graphically depicts the on-wire format of the messages exchanged during the session establishment phase, including all the pieces of information considered in Algorithms 1 and 2, as well as a set of identifiers concerning the

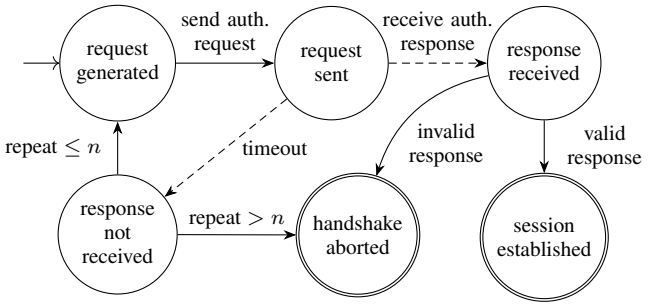

Fig. 2. State machine representing the session establishment process performed by a requester B_i, accounting for possible message losses.

actual cryptographic algorithms leveraged for the handshake. For the sake of efficiency, digital certificates are assumed to be replicated on every physical ECU when each application is initially deployed. Hence, during the handshake process, they are identified by a fingerprint, a unique identifier obtained applying a cryptographic hash function. Nonetheless, this assumption may raise concerns in case of small and resource-constrained devices with limited storage capabilities, as well as for the additional complexity introduced during the update process. In the first case, a centralized certificate repository might be leveraged to relax the storage requirements, complemented by a helper SOME/IP service exposing an interface for their retrieval and a caching mechanism to limit the overhead of the session establishment. Additionally, automatic mechanisms could be devised to transparently replicate the certificates on the different ECUs whenever the central repository is modified, thus greatly simplifying the update process. Finally, the handshake is designed to be agnostic about the transport protocol selected by SOME/IP, adopting an automatic retransmission mechanism to face message losses. To this end, the complete state machine depicting the different states that the requester can assume during the process is shown in Fig. 2.

C. Run-Time Message Protection

As soon as at least one requester B_i successfully completed the authentication handshake, the transmission of secured SOME/IP messages can start, involving both direct interactions and event notifications. Nonetheless, other session establishments might be carried out in parallel at any time, gradually granting access to a wider set of requesters. Depending on the security level characterizing the service instance considered, a different family of symmetric cryptography algorithms is leveraged to secure the messages. Yet, if a service instance is operated at *nosec* security level, unmodified messages are simply transmitted by the framework, in order to achieve full compatibility with applications built on top of vanilla SOME/IP.

In detail, the authentication and integrity of *authentication*-level messages (both the SOME/IP header and the actual payload) is guaranteed by a Message Authentication Code (MAC) appended at the end of each SOME/IP packet. Service instances operating at *confidentiality* level, on the other hand, are secured leveraging an Authenticated Encryption with Associated Data (AEAD) algorithm. It combines in a single interface both the MAC computation and the encryption of the payload containing sensible information, securely exploiting

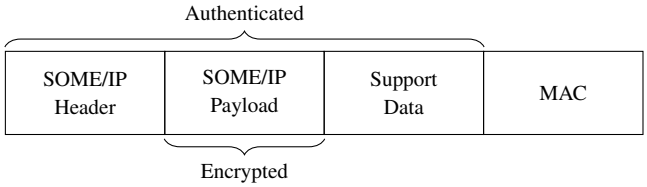

Fig. 3. High-level format of a secured SOME/IP message.

the same symmetric key. In case the security level is different from *nosec*, a sequence number $sn = \{peerid, seq\}$, composed of the peer identifier and a monotonic counter, is appended to each message to detect possible replay attempts: its integrity is effectively guaranteed by means of the MAC. Accounting for message losses and reordering typical of unreliable transport protocols (e.g. UDP), we adopted a sliding window technique, similar to the one used by Datagram Transport Layer Security (DTLS) [41]. In particular, it records the last subset of sequence numbers already received to discard the duplicates. Additionally, the sequence number is also leveraged as an initialization vector, adopted by the cryptographic algorithms to introduce a bit of randomness in the computation. Yet, some algorithms specifically require a completely random seed: in those cases, a random value would need to be generated and appended to every secured SOME/IP packet.

The high-level format of a secured message is depicted in Fig. 3. Most notably, it encompasses two main elements in addition to vanilla SOME/IP packets. First, *Support Data*, including the sequence number, the initialization vector (if different from the sequence number) and any other additional information. Second, *MAC*, corresponding to the output returned by the cryptographic algorithm and leveraged to ensure the authenticity and integrity of the entire SOME/IP message. Accounting for the constraints imposed by the SOME/IP standard on extensions, the two additional fields shall not exceed $56\,\mathrm{bytes}$ in total.

VI. Formal Verification

This section presents the main aspects of the formal verification we performed to validate the security protocol herein proposed. We start with a brief overview about why formal verification is adopted, before presenting the main modelling choices. Finally, we conclude discussing the security properties formally verified and outlining the results obtained.

A. Overview

Formal verification is a technique used to thoroughly analyze communication protocols. To this end, we leveraged it to verify the absence of logical flaws in our security protocol, i.e. to prove that no attacks are possible under certain modeling assumptions. In our analysis we exploited the Dolev-Yao formal modeling technique, and Proverif [42], an automatic cryptographic protocol verifier based on this technique. According to this approach, the attacker, which is automatically modeled by the tool, has complete control over the communication channel. Additionally, cryptography is considered ideal, focusing on the conceptual properties while abstracting away the low-level details of the actual algorithms.

Practically speaking, to use Proverif, it is necessary to formally express the cryptographic primitives adopted, as well as the behavior of the trusted actors of the protocol by means of extended pi calculus. Then, the security properties (e.g. secrecy and authentication) to be formally verified can be specified by means of queries. Sticking to an abstract approach, we decided not to model the three different security levels. Indeed, being the security level at which a service instance operates autonomously selected by the offerer depending on its security policy, a Dolev-Yao attacker would have no meaningful ways to force a downgrade. Therefore, we modeled the properties associated with the *confidentiality* level, in order to verify the effectiveness of all the considered security guarantees. In the following, we present the most relevant aspects concerning the formal verification performed to prove the correctness of the security protocol proposed. Yet, the complete model expressed in extended pi calculus is publicly available on GitHub.[3]

B. Modeling Cryptography

According to standard practice, we modeled the high-level operations associated with both symmetric and asymmetric encryption, as well as digital signatures. Most notably, depending on how rewrite rules work, the former corresponds to authenticated encryption, since decryption fails in case of tampered messages. Additionally, we introduced initialization vectors, to make two identical encrypted payloads indistinguishable by an attacker. Concerning digital certificates, we modeled them as a container binding together a public key, the identifier of a service (instances are omitted for the sake of simplicity) and a role, among *offer* and *request*. Certificate issuance is implemented through a *private* function (i.e. executable only by trusted entities) to model the impossibility for an attacker to forge digital certificates. Finally, random nonces and counters are represented in Proverif by *fresh values*, i.e. values that are initially unknown and unguessable by an attacker.

C. Trusted Protocol Actors

As attackers are automatically modeled by Proverif, we created two different processes to model the legitimate actors. They represent the behavior of the offerer and the requester during the session establishment phase, according to Algorithms 1 and 2. Additionally, we also modeled a simple run-time communication, necessary to express certain security queries (e.g. verify whether the requester correctly deciphered the symmetric key, corresponding to client authentication). In a nutshell, the requester encrypts a message using the symmetric key previously exchanged and a fresh initialization vector, sends it to the offerer that tries to decrypt it. Finally, a third process performs initialization tasks, i.e. creates the private keys and issues the digital certificates associated with the legitimate parties. Moreover, it generates a valid private key and digital certificate stating a different service and makes it available to the attacker. Hence, Proverif can verify that a given entity cannot access a specific service even if he has access to a different one. In the end, the initializer starts the execution of an unbounded number of offerers and requesters in parallel.

[3]https://github.com/netgroup-polito/secure-vsomeip

D. Security Properties

Concerning the queries formally verified by Proverif, we expressed the following: (i) Secrecy of $srvkey$: the symmetric key adopted for the run-time protection cannot be obtained or derived by the attacker. (ii) Server authentication: an attacker cannot impersonate the offerer without being detected by the requester. In other words, if a requester believes to have completed a server authentication with a given entity (i.e. **event** `end_off_auth` is triggered), then the corresponding offerer must have actually started it (i.e. **event** `begin_off_auth` must have been previously executed). Specifically, the tool verifies the injective correspondence between the two events, to require that each occurrence of the former corresponds to a distinct occurrence of the latter. (iii) Client authentication: an attacker cannot impersonate the requester and obtain $srvkey$. This query is modeled with an injective correspondence: if the offerer received a valid run-time message (i.e. **event** `end_req_auth` is executed), then a legitimate requester must have sent it (i.e. **event** `begin_req_auth` must have been previously triggered). Otherwise, an attacker would have obtained $srvkey$ and he would be able to send messages without being detected. (iv) Observational equivalence: an attacker cannot distinguish between a legitimate message and a random message, once encrypted. In other words, given two encrypted payloads, the attacker cannot discern whether they refer to the same content. This property is verified using the **choice** construct in Proverif. Yet, a positive outcome is achieved only in case symmetric cryptography is modeled explicitly adopting an initialization vector to introduce some randomness, as in actual algorithms. Conversely, a simpler model neglecting it makes the verification fail, demonstrating that an attacker would be able to recognize two identical payloads.

VII. Discussion

In this section, we discuss the most relevant advantages of our solution with respect to the direct usage of SOME/IP encapsulated within a lower secure protocol (e.g. IPSec or TLS/DTLS). Although most of the comparison refers strictly to the characteristics associated with the SOME/IP specifications, some arguments concern the compatibility with `vsomeip`. Yet, to the best of our knowledge, the GENIVI stack is the only open-source SOME/IP implementation currently available, thus being definitely worth of investigation.

The comparison focuses on the following aspects: (i) Service awareness: whether the security framework can discern between different services, to guarantee the authentication of the parties involved in the communication. (ii) Multicast support: the ability of a solution to transparently secure one-to-many communication. (iii) App-to-app security: whether real end-to-end security is guaranteed given the routing manager abstraction adopted in `vsomeip` (i.e. multi-hop communication). (iv) L4 transparency: whether security is transparently provided regardless of the L4 protocol (e.g. TCP or UDP) adopted for the actual message delivery. In other words, it indicates whether a single secure session can protect a service instance in the event that multiple transport protocols are leveraged simultaneously for different purposes. (v) IPC protection: the possibility to directly protect inter-process communications (i.e. Unix domain sockets, adopted by `vsomeip` for efficient local message delivery) without introducing the overhead of a full transport protocol.

A. SOME/IP over IPSec

Concerning SOME/IP protection, IPSec could be leveraged to establish secure channels between different ECUs hosting distributed applications. Nonetheless, although the messages transmitted across the network would be secured, the protection would be only at the network level, being IPSec unaware of the actual traffic. Hence, it would not guarantee the authentication of the parties involved in the communication (i.e. every application hosted on an ECU could send messages to another one hosted on a different ECU if a secure tunnel is established). Similarly, it would not secure the messages exchanged between applications executed on the same ECU. Finally, IPSec does not support multicast communication, forcing the SOME/IP middleware to fallback to unicast and increasing the overall bandwidth usage. Summarizing, we deem IPSec not being suitable to guarantee the level of compatibility with the SOME/IP middleware at the basis of the design of our proposal.

B. SOME/IP over TLS/DTLS

TLS is a well-established cryptographic protocol securing TCP connections, while DTLS is an extension designed to protect UDP messages. At first sight, they appear to be suitable to protect SOME/IP messages, given the promised application awareness and end-to-end security. Indeed, the feasibility of using TLS to protect in-vehicle communication has already been explored in literature [31]. Yet, in our opinion, different factors still justify a custom solution. First, being general-purpose, TLS is characterized by a fairly complex session establishment process, thus introducing a quite high overhead compared to a more targeted solution. Second, no standard version of DTLS does support multicast communication, thus limiting the efficiency in the delivery of notifications. Third, both TLS and DTLS are not suitable to protect inter-process communication unless an additional overhead is introduced (e.g. by using TCP). Finally, both security protocols cannot guarantee real end-to-end security between the sending and the receiving applications when coupled with `vsomeip`. Indeed, they would secure only the messages flowing from one central routing manager to another, leaving the local communication (i.e. between each routing manager and the actual applications executed on the same ECU) unauthenticated: thus, increasing the criticality associated with the central routing managers themselves. Similarly, since different service instances may be attached to the same $\{ip, port\}$ tuple, both TLS and DTLS could not provide real service granularity. On the other hand, our solution aims to achieve $100\,\%$ compatibility with both SOME/IP and `vsomeip`, while featuring a more efficient session establishment procedure.

VIII. Experimental Evaluation

To experimentally evaluate the performance of the solution proposed to secure SOME/IP communication, we integrated

the security framework within `vsomeip`. Nonetheless, we focused on the implementation of the security protocol, leaving the usage of a secure storage as a future work. Concerning cryptography, we leveraged OpenSSL, being it a well-known, efficient and commercial grade library available for a wide range of systems and architectures. The complete source code of the proof of concept is publicly available on GitHub.[4] Part of the results described in the remainder of this section stem from evaluation metrics originally presented in [17].

The testbed adopted for the experimental evaluation was composed of two identical NXP's development boards, based on the i.MX 7Dual applications processor. They encompass two Arm Cortex-A7 cores operating at up to $1\,\mathrm{GHz}$ and $1\,\mathrm{GB}$ of DDR3 RAM. We directly interconnected the two ECUs through an Ethernet cable, while limiting the speed to $100\,\mathrm{Mbps}$ to match the current standard in automotive environments. Concerning the operating system, we adopted an embedded Posix Linux distribution, according to the Adaptive Autosar guidelines. Overall, although not having used real production ECUs due to licensing issues, we deem the whole setup matches the features of a real in-vehicle deployment. Indeed, both SOME/IP and the whole Adaptive Autosar standard have been developed with relatively high-performance microprocessor-based ECUs in mind, focusing on use cases such as infotainment, connected vehicle and ADAS.

A. Session Establishment

The first target of the benchmark evaluation was the session establishment phase, to measure the latency introduced before a service could be accessed. To this end, we developed two applications: the former acts as a server, offering 256 different SOME/IP services. The latter, on the other hand, concurrently requests a varying number of services (i.e. implicitly starts the corresponding session establishment), to evaluate how the handshake phase scales. Technically speaking, all authentication handshakes have been performed over UDP, adopting RSA-2048 as a strong asymmetric cryptography algorithm for encryption and digital signatures. Fig. 4 presents the outcome of the evaluation when varying the amount of parallel session establishments. Considering the dotted line as a reference, the overall trend confirms the scalability of the approach proposed. Indeed, doubling the number of services resulted in an increase of less than double the total authentication overhead, thanks to the efficient exploitation of the available parallelism.

B. Run-Time Protection

Subsequently, we studied the effects of the three available security levels on the transmission of actual SOME/IP packets, while adopting vanilla `vsomeip` as a reference. Concerning cryptography, we selected ChaCha20-Poly1305 as symmetric algorithm to secure both *authentication* and *confidentiality*-level services, given its outstanding performance even without using hardware accelerators. To account for multiple use cases and evaluate the different facets associated with the communication, we adopted three complementary approaches.

[4]https://github.com/netgroup-polito/secure-vsomeip

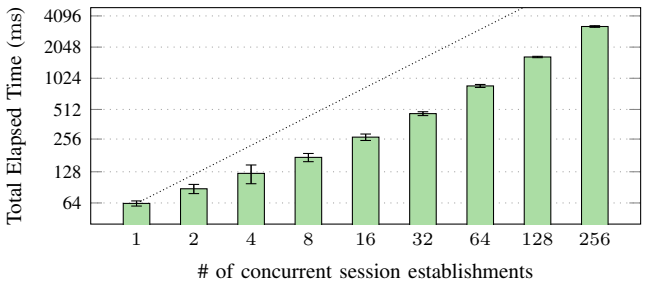

Fig. 4. Evaluation of the time required to concurrently complete multiple session establishments varying the handshake parallelism (the dotted line represents a reference corresponding to a doubling in the elapsed time).

Namely, we considered the effects on message round-trip-time (RTT), number of interactions per second (IPS) and throughput. Additionally, concerning overall performance, we complemented the benchmarks with CPU usage measurements.

C. Round Trip Time

First, we evaluated the transmission of periodic messages in absence of other traffic, assessing the latency introduced by message authentication and confidentiality. To this end, we measured the total time required for an application to send a request message and receive its corresponding response. Hence, it accounts for the computations performed by the middleware as well as the network overhead. The benchmark has been repeated considering multiple payload sizes, ranging from 1 to $1024\,\mathrm{bytes}$, to evaluate how much it influences the results. Only the request was modified between different runs, while the responses were characterized by the absence of the payload.

Concerning the topology, both applications have been initially executed on the same device. Thus, `vsomeip` leveraged Unix domain sockets for the delivery of the packets and the CPU was actually shared by both parties, halving the total computational capacity. Yet, this configuration is deemed to better highlight the effects ascribable to security, being the network latency much smaller compared to a physical network. Fig. 5a presents the results obtained, which elicit three main considerations. First, security obviously introduced an overhead in the message transmission, accounting for about $25\,\%$ the total RTT. Yet, *authentication* and *confidentiality* security levels appear to be characterized by almost the same results. Second, no significant differences emerge varying the message size, with the security overhead becoming a little more prominent only for the biggest payload. Third, the CPU usage was quite high during the entire evaluation: indeed, both applications were executed on the same ECU, effectively doubling the values presented. Yet, it remained constant, presenting no valuable deviations when increasing the security level.

Second, we hosted the two applications on different ECUs, to assess the overhead in case of actual network communication. In this configuration, the measured RTTs were almost one order of magnitude higher than the previous situation ($\approx 2.5\,\mathrm{ms}$), due to physical network communication overhead. Conversely, the effects of security appeared to be way less prominent, playing only a very small role in the total values. Similarly, the CPU load remained definitely low ($< 5\,\%$), being at the same time only slightly influenced by cryptography.

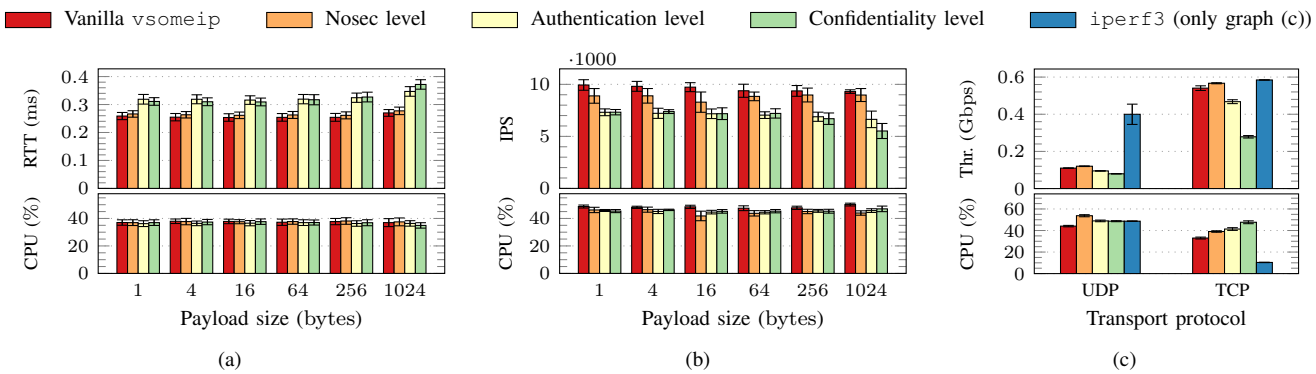

Fig. 5. Outcome of the benchmarks comparing vanilla vsomeip and our security-enhanced version in terms of (a) round-trip-time (RTT), in case of local communication, (b) interactions-per-second (IPS), in case of local communication and (c) throughput, in case of remote communication.

D. Interactions Per Second

Then, we assessed the performance of secure SOME/IP under stress, measuring the number of maximum interactions per second (i.e. requests that an offerer can reply to in a given unit of time). Although adopting a methodology similar to the RTT benchmarks, the client was now configured to send the highest possible number of messages in parallel. Fig. 5b shows the outcome of the evaluation concerning local communication. Generally speaking, security features impacted for about $25-30\,\%$, without particular differences when varying the payload size or moving from the *authentication* to the *confidentiality* security level. Furthermore, the results were definitely similar regardless of the type of communication adopted (i.e. local or remote), being the network overhead mitigated by the high number of parallel requests. Additionally, both benchmarks significantly overloaded the CPU. Indeed, in case of local communication, both the offerer and the requester were concurrently hosted by the same ECU, thus halving the total computational capacity and corresponding to a total CPU usage slightly below $100\,\%$. As for the remote communication benchmark, instead, the CPU load remained consistently around $75\,\%$. Nonetheless, the introduction of security did not cause significant differences in the CPU load, being all measurements contained in the same uncertainty band.

E. Throughput

Moving on, we evaluated the performance concerning the transmission of high amounts of data (e.g. video streams). To this end, we leveraged the publish/subscribe communication pattern: the offerer continuously streams SOME/IP packets, eventually consumed by one of more requesters. Regarding the transport protocol, we assessed both the usage of UDP, given its support to multicast communication, and TCP, which is deemed to be more suitable for the transmission of large chunks of data. As for the former, we configured the payload to contain the maximum possible amount of data ($1400\,\text{bytes}$, considering the SOME/IP limitations). Messages transported over TCP, on the other hand, were characterized by $64\,\text{kB}$ payloads. To better assess the actual throughput, we also relaxed the limitation regarding the network speed. Hence, during this benchmark, the two ECUs were effectively interconnected at $1\,\text{Gbps}$.

Fig. 5c presents the outcome of the evaluation, measuring both the ability of the sender to transmit the messages, and the capacity of the receiver to consume them (i.e. including the necessary authentication and decryption operations). Along with the other values, the iperf3 bar represents a reference measurement obtained using the well-known tool. It aims to measure the bare network performance on top of UDP and TCP sockets, in case the SOME/IP abstraction was not adopted. At a glance, it clearly stands out the significantly better performance associated with iperf3 compared to vsomeip. Indeed, in case UDP is adopted, the former achieved a throughput almost four times higher, while comparably loading the CPU. Conversely, with regards to the TCP benchmark, similar throughput values were obtained; yet, a fourfold drop in CPU usage was observed when using iperf3. This comparison points out the high overhead introduced by the vsomeip implementation. Hence, it clearly remains a significant room for improvements, which security features would undoubtedly also benefit from. In this respect, focusing on the UDP benchmark, *authentication* and *confidentiality* levels respectively introduced an additional slow down accounting for $15\,\%$ and $25\,\%$. When adopting TCP as a transport protocol, on the other hand, confidentiality impacted significantly on the effective throughput, being responsible for a $50\,\%$ drop in performance compared to vanilla vsomeip.

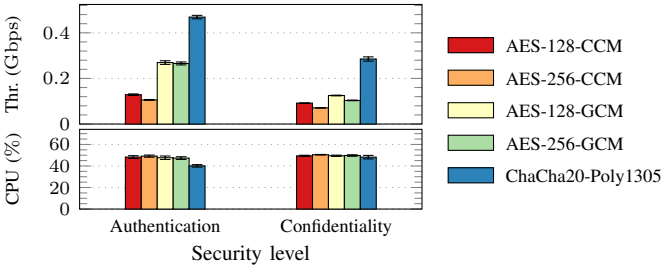

Fig. 6. Throughput (over TCP) and CPU usage comparison between different symmetric cryptography algorithms adopted to secure SOME/IP messages.

Finally, Fig. 6 compares the throughput values achieved when adopting different, state-of-the-art, authenticated encryption algorithms. In detail, the evaluations were carried out over TCP, operating the cryptographic algorithms both in authentication-only and AEAD mode (i.e. *confidentiality*-level). ChaCha20-Poly1305 clearly emerged as the fastest alternative. Indeed, although offering a $256\,\text{bits}$ security level, it achieved twice as much throughput compared to the second, namely AES-GCM. AES-CCM, on the other hand, appeared to be the slowest,

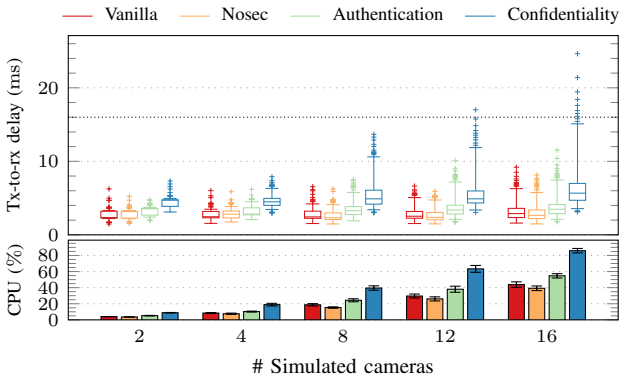

Fig. 7. Performance comparison between vanilla `vsomeip` and our security-enhanced version in terms of tx-to-rx delay and CPU usage, when simulating multiple cameras operating at 60 fps and generating compressed video frames of 42.5 kB [43], in case of remote communication (the dotted line represents a reference corresponding to a delay equal to 16 ms).

especially concerning *authentication*-level services. As for the AES security level, about a 20 % additional overhead was introduced when moving from 128 bits to 256 bits keys.

F. Realistic Scenario

The previous benchmarks were meant to independently measure the different facets of the overhead introduced by the run-time protection. Yet, to generalize the overall evaluation and assess the actual effects in a broader and more realistic scenario, we then focused on simulating an ADAS system. Specifically, we simulated multiple cameras in parallel, each one operating at 60 fps and generating compressed video frames of 42.5 kB (≈ 21 Mbps per camera) [43]. All frames were generated and timestamped by the first board, and sent to the second one (over a 1 Gbps link and adopting TCP as a transport protocol) to evaluate the tx-to-rx delay and the CPU usage when varying the number of cameras and increasing the security level. To this end, we set 16 ms as an upper bound for the delay, to guarantee the reception of a frame before the generation of the subsequent. Yet, stricter requirements may need to be enforced depending on the specific application and the amount of processing to be performed on the data received. Finally, as for clock synchronization, we leveraged the Precision Time Protocol, which delivers sub-microsecond accuracy on LANs [44].

Fig. 7 summarizes the results of the evaluation, considering the number of parallel cameras ranging from 2 (≈ 42 Mbps) to 16 (≈ 336 Mbps). Concerning the tx-to-rx delay, the box plot represents the extension of the quartiles, as well as of the 1st and the 99th percentile (a representative set of outliers is shown as individual points) for 100 s long simulations. Focusing first on the reference scenarios (i.e. those with no security features enabled), the median delay remained confined in the 2.5 – 3 ms range. Yet, an increase in the number of cameras corresponded to a higher dispersion, as well as a proportional growth in CPU usage. The introduction of *authentication*-level security caused a 15 – 25 % additional overhead, with a more significant increase for what regards the 99th percentile mark. Nonetheless, all frames but a very few outliers required less than 10 ms to be correctly received and authenticated,

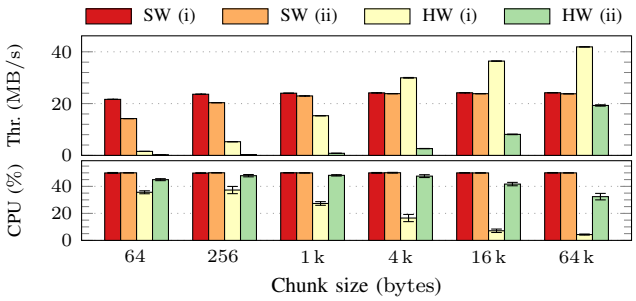

Fig. 8. Performance comparison between software and hardware implementations of AES-128-CBC, when encrypting multiple chunks of data within (i) the same and (ii) different cryptographic contexts.

hence completely satisfying the requirements previously set. Conversely, *confidentiality* was associated with a considerably higher impact, both in terms of delay (median and 99th percentile), as well as CPU usage. Additionally, as for the most demanding case, a few frames failed to be delivered within the expected deadline.

We believe the outcome of this benchmark to be twofold. One the one hand, it showed the sustainability of the security features, having the requirements been satisfied in all but the most demanding scenario. On the other hand, it uncovered the much higher overhead associated with confidentiality, compared to authentication only. Hence, requiring a careful analysis to assess whether for each service this additional protection is actually required, to avoid introducing unnecessary overhead. Nonetheless, it is also fair to mention that the ECUs leveraged for infotainment and ADAS systems on modern vehicles are much more powerful than our testbed (e.g. the NVIDIA Drive PX family features up to 16 ARM cores), thus strongly flattening the possible impact of the security features.

G. Hardware Acceleration

For the sake of generality, the performance evaluation has been completed adopting software cryptographic implementations only. However, typical automotive ECUs may be complemented with hardware accelerators. Hence, being the i.MX7 family of boards characterized by a Cryptographic Accelerator and Assurance Module (CAAM), we investigated also the performance achievable when leveraging offloading features. To this end, we exploited the `cryptodev-linux` kernel module[5], which provides an abstraction to allow userspace applications (e.g. OpenSSL) to transparently benefit from hardware accelerators. Focusing on symmetric cryptography, we managed to offload AES-CBC (encryption) and HMAC-SHA1 (authentication), along with other legacy algorithms. Yet, no authenticated encryption support seemed to be provided.

Fig. 8 presents an evaluation targeting AES-128-CBC, which compares the performance of the software and the hardware implementations in terms of throughput (i.e. amount of data encrypted per second) and CPU usage.[6] Two different scenarios have been considered: (i) the encryption of a stream of chunks

[5]http://cryptodev-linux.org

[6]Differently from the previous benchmarks, this evaluation has been performed independently from the SOME/IP middleware, to better highlight the performance differences of the cryptographic algorithms.

of data (e.g. a TCP flow) using the same cryptographic context (i.e. (key, iv) pair); (ii) the encryption of independent messages, such as those generated by SOME/IP: hence, the context is reinitialized before every operation (i.e. varying the iv), to avoid requiring a strict ordering during the reception (e.g. if messages are transmitted over UDP). Focusing first on the results using the software implementation, no relevant differences emerged when varying either the size of the chunks to be encrypted or the scenario considered. Indeed, the throughput remained stable around $24\,\mathrm{MB/s}$, with $100\,\%$ single core CPU usage. The outcome of the measurements leveraging the hardware offloading capabilities, on the other hand, turned out to be strongly influenced by the amount of data to be encrypted. Yet, this dependency could be easily explained by the high overhead required to switch from user to kernel space and interact with the hardware module, which pays off only with larger blocks (both in terms of throughput and CPU usage, with the latter going as low as $5\,\%$ in the best case). Additionally, the reinitialization of the cryptographic context (i.e. scenario (ii)) proved to cause serious performance drops, loosing all the advantages associated with hardware acceleration.

This preliminary evaluation pointed out some limitations of the hardware module featured by the testbed, suggesting the adoption of software implementations in our PoC and for the benchmarks. Indeed, it provides limited support for state-of-the-art cryptographic algorithms (e.g. authenticated encryption ones), as well as it is characterized by unsatisfactory performance, especially in case of small messages. Indeed, ChaCha20-Poly1305 consistently achieved better results, even though it combines both encryption and authentication in a single interface and offers a $256\,\mathrm{bits}$ security level. Although bound to the specificity of the platforms and the abstractions adopted, we believe these measurements can shed some light on the possible limitations of hardware acceleration.

IX. Conclusions and Future Work

Novel in-vehicle services using high-speed communications are being increasingly implemented. Yet, over the last decade, many researchers have demonstrated that unsecured network messages pose a serious threat in the vehicular domain, perhaps granting a malicious entity the control of safety-critical systems and endangering the life of the passengers. Starting from these considerations, we proposed a novel framework to secure SOME/IP, an emerging middleware designed by the AUTOSAR consortium to account for future-proof scenarios. Differently from the encapsulation of SOME/IP messages within a lower secure protocol, our solution aims to introduce no limitations in the functionalities provided by the middleware, as well as being $100\,\%$ compatible with its open-source implementation `vsomeip`. Simple, high-level rules are leveraged to allow automotive companies to specify the set of peers each application is authorized to talk to, sticking to the service abstraction adopted by SOME/IP. In addition, three incremental security levels are made available, in order to associate each service to the one best matching its requirements. The security protocol at the core of our solution has been formally modeled and verified, so providing high assurance that the expected security properties hold. Finally, extensive experimental evaluations confirmed the introduction of a fairly limited overhead during both the establishment of secure sessions and the run-time protection. Nonetheless, most of the burden is ascribable to cryptography: better performance could be achieved using a weaker yet faster algorithm or leveraging more powerful hardware accelerators to relieve the main CPU from the task. Future work will focus on preventing DoS attacks, aiming to overload the devices with an excessive amount of traffic to prevent legitimate communications. Additionally, a thorough study will be carried out to understand how to integrate the security protocol with hardware security, to preclude sufficiently motivated attackers from tampering with cryptographic material.

Acknowledgment

We thank Paolo Montuschi for the precious suggestions he gave us to improve the paper.

References

[1] R. N. Charette, "This car runs on code," *IEEE spectrum*, vol. 46, no. 3, p. 3, Feb. 2009.

[2] O. Burkacky, J. Deichmann, G. Doll, and C. Knochenhauer, "Rethinking car software and electronics architecture," *McKinsey & Co.*, Feb. 2018.

[3] H. Winner, S. Hakuli, F. Lotz, and C. Singer, *Handbook of driver assistance systems: Basic information, components and systems for active safety and comfort*. Springer, 2015.

[4] G. Meixner and C. Müller, *Automotive User Interfaces: Creating Interactive Experiences in the Car*. Springer, 2017.

[5] M. Broy, "Challenges in automotive software engineering," in *Proc. 28th ACM Int. Conf. Software Engineering*, May 2006, pp. 33–42.

[6] K. H. Johansson, M. Törngren, and L. Nielsen, "Vehicle applications of controller area network," in *Handbook of networked and embedded control systems*. Springer, 2005, pp. 741–765.

[7] IEEE, "IEEE standard for Ethernet amendment 1: Physical layer specifications and management parameters for 100 Mb/s operation over a single balanced twisted pair cable (100BASE-T1)," *IEEE Std 802.3bw-2015 (Amendment to IEEE Std 802.3-2015)*, pp. 1–88, Mar. 2016.

[8] L. L. Bello, "The case for Ethernet in automotive communications," *SIGBED Rev.*, vol. 8, no. 4, pp. 7–15, Dec. 2011.

[9] S. Tuohy, M. Glavin, C. Hughes, E. Jones, M. Trivedi, and L. Kilmartin, "Intra-vehicle networks: A review," *IEEE Trans. Intell. Transp. Syst.*, vol. 16, no. 2, pp. 534–545, Apr. 2015.

[10] AUTOSAR, *SOME/IP Protocol Specification*, 2016. [Online]. Available: https://www.autosar.org/fileadmin/user_upload/standards/foundation/1-0/AUTOSAR_PRS_SOMEIPProtocol.pdf

[11] J. Ning, J. Wang, J. Liu, and N. Kato, "Attacker identification and intrusion detection for in-vehicle networks," *IEEE Commun. Lett.*, vol. 23, no. 11, pp. 1927–1930, Nov. 2019.

[12] T. Hoppe, S. Kiltz, and J. Dittmann, "Security threats to automotive CAN networks — Practical examples and selected short-term countermeasures," *Rel. Eng. Syst. Safety*, vol. 96, no. 1, pp. 11–25, Jan. 2011.

[13] K. Koscher *et al.*, "Experimental security analysis of a modern automobile," in *Proc. IEEE Symp. Security and Privacy*, May 2010, pp. 447–462.

[14] S. Checkoway *et al.*, "Comprehensive experimental analyses of automotive attack surfaces," in *Proc. 20th USENIX Conf. Security*, Aug. 2011, pp. 77–92.

[15] C. Miller and C. Valasek, "Remote exploitation of an unaltered passenger vehicle," *Proc. Black Hat USA*, Aug. 2015.

[16] S. Nie, L. Liu, and Y. Du, "Free-fall: hacking tesla from wireless to CAN bus," *Proc. Black Hat USA*, Jul. 2017.

[17] M. Iorio, A. Buttiglieri, M. Reineri, F. Risso, R. Sisto, and F. Valenza, "Protecting in-vehicle services: Security-enabled SOME/IP middleware," *IEEE Veh. Technol. Mag.*, vol. 15, no. 3, pp. 77–85, Sep. 2020.

[18] B. Groza, S. Murvay, A. van Herrewege, and I. Verbauwhede, "LiBrA-CAN: A lightweight broadcast authentication protocol for controller area networks," in *Proc. Int. Conf. Cryptology and Network Security*, Dec. 2012, pp. 185–200.

[19] P. Murvay and B. Groza, "Source identification using signal characteristics in controller area networks," *IEEE Signal Process. Lett.*, vol. 21, no. 4, pp. 395–399, Apr. 2014.

[20] Q. Wang and S. Sawhney, "Vecure: A practical security framework to protect the CAN bus of vehicles," in *Proc. Int. Conf. Internet of Things (IOT)*, Oct. 2014, pp. 13–18.

[21] J. H. Kim, S. Seo, N. Hai, B. M. Cheon, Y. S. Lee, and J. W. Jeon, "Gateway framework for in-vehicle networks based on CAN, FlexRay, and Ethernet," *IEEE Trans. Veh. Technol.*, vol. 64, no. 10, pp. 4472–4486, Oct. 2015.

[22] E. Wang, W. Xu, S. Sastry, S. Liu, and K. Zeng, "Hardware module-based message authentication in intra-vehicle networks," in *Proc. ACM/IEEE 8th Int. Conf. Cyber-Physical Systems*, Apr. 2017, pp. 207–216.

[23] K.-T. Cho and K. G. Shin, "Fingerprinting electronic control units for vehicle intrusion detection," in *Proc. 25th USENIX Conf. Security Symp.*, Aug. 2016, pp. 911–927.

[24] S. Jain and J. Guajardo, "Physical layer group key agreement for automotive controller area networks," in *Proc. Int. Conf. Cryptographic Hardware and Embedded Systems*, Aug. 2016, pp. 85–105.

[25] S. Nürnberger and C. Rossow, "– vatiCAN – vetted, authenticated CAN bus," in *Proc. Int. Conf. Cryptographic Hardware and Embedded Systems*, Aug. 2016, pp. 106–124.

[26] B. Groza and P. Murvay, "Security solutions for the controller area network: Bringing authentication to in-vehicle networks," *IEEE Veh. Technol. Mag.*, vol. 13, no. 1, pp. 40–47, Mar. 2018.

[27] D. Püllen, N. A. Anagnostopoulos, T. Arul, and S. Katzenbeisser, "Using implicit certification to efficiently establish authenticated group keys for in-vehicle networks," in *Proc. IEEE Vehicular Networking Conf. (VNC)*, Dec. 2019, pp. 168–175.

[28] A. Perrig, R. Canetti, J. D. Tygar, and Dawn Song, "Efficient authentication and signing of multicast streams over lossy channels," in *Proc. IEEE Symp. Security and Privacy*, May 2000, pp. 56–73.

[29] M. Wolf, A. Weimerskirch, and C. Paar, "Secure in-vehicle communication," in *Embedded Security in Cars: Securing Current and Future Automotive IT Applications*. Springer, 2006, pp. 95–109.

[30] M. Hamad, M. Nolte, and V. Prevelakis, "A framework for policy based secure intra vehicle communication," in *Proc. IEEE Vehicular Networking Conf. (VNC)*, Nov. 2017, pp. 1–8.

[31] D. Zelle, C. Krauß, H. Strauß, and K. Schmidt, "On using TLS to secure in-vehicle networks," in *Proc. 12th Int. Conf. Availability, Reliability and Security*, Aug. 2017, pp. 1–10.

[32] M. Oertel and B. Zimmer, "E/E architectures with AUTOSAR adaptive," Vector, Tech. Rep., May 2019.

[33] AUTOSAR, *Explanation of Adaptive Platform Design*, 2017. [Online]. Available: https://www.autosar.org/fileadmin/user_upload/standards/adaptive/17-10/AUTOSAR_EXP_PlatformDesign.pdf

[34] ——, *SOME/IP Service Discovery Protocol Specification*, 2017. [Online]. Available: https://www.autosar.org/fileadmin/user_upload/standards/foundation/1-3/AUTOSAR_PRS_SOMEIPServiceDiscoveryProtocol.pdf

[35] G. Pardo-Castellote, "OMG data-distribution service: architectural overview," in *Proc. 23rd Int. Conf. on Distributed Computing Systems Workshops*, May 2003, pp. 200–206.

[36] Object Management Group (OMG), *Remote Procedure Call over DDS (DDS-RPC) (Version 1.0)*, 2017. [Online]. Available: https://www.omg.org/spec/DDS-RPC/1.0/PDF

[37] ——, *The Real-time Publish-Subscribe Protocol DDS Interoperability Wire Protocol (DDSI-RTPS) Specification (Version 2.3)*, 2019. [Online]. Available: https://www.omg.org/spec/DDSI-RTPS/2.3/PDF

[38] ——, *DDS Security (Version 1.1)*, 2018. [Online]. Available: https://www.omg.org/spec/DDS-SECURITY/1.1/PDF

[39] M. Wolf and T. Gendrullis, "Design, implementation, and evaluation of a vehicular hardware security module," in *Proc. Int. Conf. Information Security and Cryptology*, Dec. 2012, pp. 302–318.

[40] D. Dolev and A. Yao, "On the security of public key protocols," *IEEE Trans. Inf. Theory*, vol. 29, no. 2, pp. 198–208, Mar. 1983.

[41] E. Rescorla and N. Modadugu, "Datagram transport layer security version 1.2," RFC Editor, RFC 6347, 2012.

[42] B. Blanchet, "An efficient cryptographic protocol verifier based on prolog rules," in *Proc. 14th IEEE Computer Security Foundations Workshop*, Jun. 2001, pp. 82–96.

[43] J. Migge, J. Villanueva, N. Navet, and M. Boyer, "Insights on the performance and configuration of AVB and TSN in automotive ethernet networks," in *Proc. 9th Eur. Congr. Embedded Real Time Software and Systems (ERTS 2018)*, Jan. 2018, pp. 1–14.

[44] IEEE, "IEEE standard for a precision clock synchronization protocol for networked measurement and control systems," *IEEE Std 1588-2019 (Revision of IEEE Std 1588-2008)*, pp. 1–499, Jun. 2020.

Marco Iorio received the M.Sc. degree in computer engineering from the Politecnico di Torino, Italy, in 2018, where he is currently pursuing the Ph.D. degree. His research interests include cooperative driving, vehicular networks and cybersecurity.

Massimo Reineri received his Ph.D. in electronics and communication engineering from the Politecnico di Torino, Italy, in 2013. In 2013, Massimo joined the "Istituto Superiore Mario Boella" as post-doctorate researcher. In 2014, he moved to Magneti Marelli Motorsport as R&D Telemetry Engineer for Formula 1 Championship. In 2017, Massimo joined Italdesign, where he is currently working as Connected Car Specialist. His main activities focus on testbed, content downloading, communication protocols and user applications for Vehicular Networks.

Fulvio Risso received the M.Sc. (1995) and Ph.D. (2000) in computer engineering from the Politecnico di Torino, Italy. He is currently Associate Professor with the Politecnico di Torino. His research interests focus on high-speed and flexible network processing, edge/fog computing, software-defined networks, network functions virtualization. He has co-authored more than 130 scientific papers.

Riccardo Sisto received the Ph.D. degree in computer engineering from the Politecnico di Torino, Italy, in 1992. Since 2004, he has been a Full Professor of computer engineering with the Politecnico di Torino. He has authored and coauthored more than 120 scientific papers. His main research interests include formal methods, applied to distributed software and communication protocol engineering, distributed systems, and computer security. He is a Senior Member of the ACM.

Fulvio Valenza received the M.Sc. (summa cum laude) in 2013 and the Ph.D. (summa cum laude) in Computer Engineering in 2017 from the Politecnico di Torino, Italy. His research activity focuses on network security policies. Currently he is a Researcher at the Politecnico di Torino, Italy, where he works on orchestration and management of network security functions in the context of SDN/NFV-based networks.