Protecting In-Vehicle Services with Security-Enabled SOME/IP*

Marco Iorio*, Alberto Buttiglieri†, Massimo Reineri†, Fulvio Risso*, Riccardo Sisto*, Fulvio Valenza*
*Politecnico di Torino, Torino, Italy
{name.surname}@polito.it
†Italdesign, Moncalieri, Italy
{name.surname}@italdesign.it

VEHICLES are becoming every generation smarter and more ICT-oriented. However, computerization is posing unforeseen challenges in a sector where the first goal must be safety: car hacking has been shown to be a real threat. This paper presents a novel mechanism to provide improved security to the applications that are executed in the vehicle, based on the principle of defining exactly who can talk to whom. The proposed security framework targets Ethernet-based communications and it is tightly integrated within the emerging SOME/IP middleware. No complex configurations are needed: simple high-level rules, clearly stating the communications allowed, are the only element required to enable the security features. The solution designed has been implemented as a PoC inside the `vsomeip` stack to evaluate the validity of the approach proposed: experimental measurements confirm that the additional overhead introduced in end-to-end communication is negligible.

I. Introduction

Modern cars are characterized by dozens of different Electronic Control Units (ECUs), each one hosting one or more applications devoted to monitor and manage every single aspect of the vehicle itself. Advanced Driving Assistance Systems are going further, moving the control of safety critical systems, like braking and steering, to computers, algorithms, and software.

Different devices cannot operate in complete isolation: communications and protocols are fundamental to allow the exchange of information within the car. To this extent, Controller Area Network (CAN bus) is a well-established technology forming the backbone of every in-vehicle network, being it suitable for strongly real-time oriented applications. By its side, Automotive Ethernet [1] is gaining more and more importance, enabling high-bandwidth communications and replacing a plethora of complex proprietary technologies [2]. On the top of bare Ethernet, Service Oriented Architectures (SOAs) are becoming increasingly popular as a high-level abstraction to support complex applications and allow for maximum flexibility [3]. According to this design pattern, a system is composed of a set of services providing different functionalities. Additionally, the communication is abstracted through a virtual bus, allowing the exchange of messages regardless of the physical device where each service is currently executed on. To date, one of the most promising SOA middleware for in-vehicle communications is SOME/IP [4], having it been designed explicitly for automotive use-cases by the AUTOSAR consortium.

With the increasing prominence of computer-based systems, new challenges are threatening the life of millions of unaware drivers. During recent years, different researchers have succeeded in exploiting specifically crafted network messages to take over the control of safety-critical systems [5], [6]. Albeit the presented attacks do require physical access to in-vehicle buses, isolation cannot be assumed as a sufficient prevention. On the one hand, [7], [8] demonstrated that many commercial vehicles present serious vulnerabilities in network stack implementations, namely Bluetooth, Wi-Fi and 4G, granting wicked individuals the possibility to remotely access the cars' internals. On the other one, possible attacks may originate from within the vehicle itself: both unauthorized devices and compromised software updates could be installed by dishonest mechanics or for tuning reasons.

Being a vital system in every vehicle, the CAN bus has already been the subject of extensive research to increase the security and prevent malicious attacks. Groza et al. [9] recently presented a survey of a wide range of possible solutions, including both active protections, which exploit cryptographic functions to guarantee message authentication, and physical layer solutions, using signal patterns to distinguish between different nodes. On the other hand, much less attention has been devoted to the protection of Ethernet-based communications in the automotive domain. Although effective and mature protocol suites are available in the broader field of ICT networking, they appear not to fit well the peculiarities of in-vehicle networks and especially of the SOME/IP middleware. The IPsec protocol, for example, is restricted by limited granularity due to application unawareness, while TLS does not support multicast communications and requires a rather complex authentication handshake. Hamad et al. [10] recently proposed a framework to provide secure communications between ECUs by exploiting security policies. However, this solution requires the definition of low-level rules, strongly limiting the dynamism introduced by SOAs.

This paper presents a novel security framework protecting SOME/IP-based communications. It has been designed with simplicity in mind, leveraging the integration within the communication middleware to replace low-level policies with

*Patent Pending

simple high-level rules, clearly stating the permitted traffic matrix in terms of services. Along with the definition of the allowed communications, the proposed solution grants application developers the capability of assigning a different security level to each service. Thus, the protection can be tuned depending on the specific level of criticality, to achieve the best trade-off between security and performance.

II. SOME/IP

SOME/IP is an emerging communication middleware standardized by AUTOSAR. It aims to include all the features required by automotive, Ethernet-oriented, use-cases while fulfilling the hard requirements regarding resource consumption in vehicles. The middleware is designed to provide a service-oriented abstraction on the top of one or more transport protocols, mainly UDP and TCP, and offers two communication patterns: request/response and publish/subscribe. The former corresponds to standard Remote Procedure Call (RPC), providing the possibility to invoke functions made available by other applications. The latter, on the other hand, exploits notifications managed directly by the middleware to decouple the sender from the recipients and allow for traffic optimization. Additionally, one of the most noteworthy features provided by SOME/IP is the service discovery [11], which can dynamically advertise the availability of different services as well as manage the subscription to selected events.

However, albeit being deemed to be very promising as a communication middleware, SOME/IP does not include any security functionality, leaving the applications and the messages transmitted across the network completely unprotected from malicious attacks.

A. vsomeip

The `vsomeip` stack[1] is an open-source implementation of the SOME/IP specifications, designed as part of the GENIVI project. A representation of the architecture proposed by `vsomeip` is depicted in Fig. 1, which shows two ECUs interconnected through an Ethernet link. Different applications are being executed at the same time on the top of a Linux kernel. Each one is characterized by its own instance of the `vsomeip` library, which can be further subdivided into two main building blocks.

The upper part shows the module providing the public API, which is exploited by all applications for the interaction with the library itself. The core part of `vsomeip` is the *routing manager* entity, responsible for message delivery both locally and remotely. Two different types of the routing manager do coexist and are completely transparent from the applications' point of view:

- *routing manager*, the full-fledged version loaded by only one application per each device. It is responsible for sending messages to and receiving them from applications residing on remote devices, by managing the transport endpoints (i.e. the TCP and UDP sockets). Furthermore, it is in charge of loading the service discovery (if enabled).

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

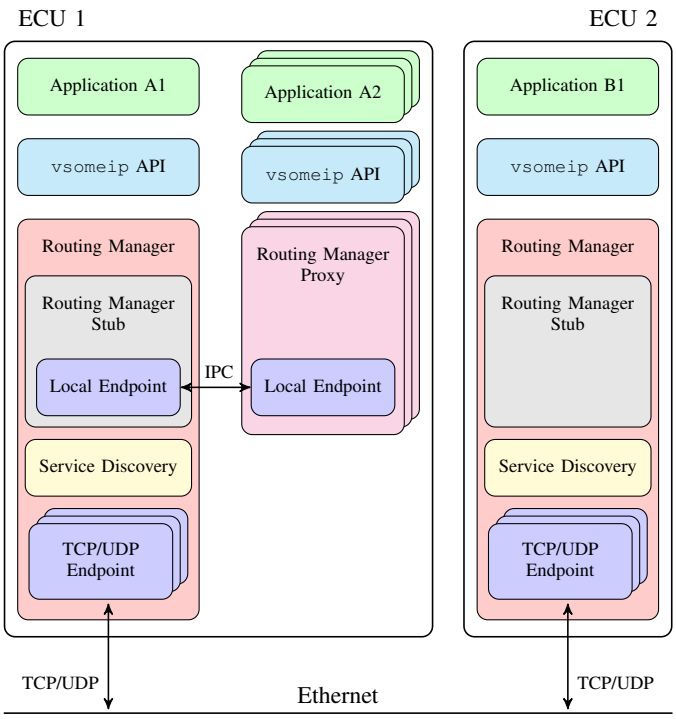

Fig. 1. `vsomeip` architecture, freely redrawn from [12].

- *routing manager proxy*, executed by all the other instances and in charge of the communications with other applications residing on the same ECU, leveraging Unix domain sockets, while messages toward remote recipients are redirected to the main routing manager.

According to the `vsomeip` documentation, the framework includes some security features based on Unix credentials, thus available only for local communications. Nonetheless, these capabilities appear to be somehow limited and weak, due to the total lack of protection for what regards communications between remote devices and the usage of unauthenticated configuration files.

III. Securing SOME/IP

Our security framework, aiming to protect SOME/IP communications, has been designed to operate at service instance granularity. Hence, each instance of a SOME/IP service is considered as a unique entity to which a specific application can be either allowed or denied access. Nonetheless, being services just logical abstractions grouping together methods and events, the final security granularity depends on the architectural decisions made by application developers.

A different security level, among the three incremental alternatives enumerated in the following, can be assigned to each service instance: thus, it is possible to select for each service the best balance between protection and computational/network overhead.

1) *Nosec*, corresponding to vanilla SOME/IP. Although not providing any security property, it may be suitable for very simple and low-criticality services, since it introduces no additional complexity to the transmission.

2) *Authentication*, assuring that only allowed applications can send messages associated to a specific service: in other words, it attests data authentication and integrity. Additionally, this security level also prevents replay attacks, characterized by the capture of valid packets for a subsequent retransmission to trigger again the same action. These security properties are deemed to be fundamental in vehicular networks. Indeed, while data exchanged may not need to remain secret, it is of the highest importance that every application processes the information received only when its authenticity and integrity is assured. Otherwise, malicious messages injected in the network could succeed in fooling the application logic, possibly triggering safety-critical physical actions at arbitrary time intervals.
3) *Confidentiality*, guaranteeing all the security properties offered by the *authentication* level and, additionally, data confidentiality, to preclude unauthorized parties from accessing the information exchanged. Hence, although being named *confidentiality*, this security level combines all the three main security properties that can be associated to messages: authentication, integrity and confidentiality. Confidentiality, alone, would not be able to provide sufficient security: an attacker, in fact, would remain able to inject malicious messages undetectable by the receivers, albeit with only unpredictable payloads. Even though currently not characterized by the same importance of the other security properties, the introduction of confidentiality might be useful also in in-vehicle networks. In particular, application developers could assign this security level to prevent unauthorized aftermarket ECUs from reading the messages exchanged on the bus, thus selectively limiting their ability to grab external information. Additionally, it may contribute to the protection of intellectual property, avoiding that the content of network messages provides hints about complex application logic.

A. Security protocol

The core of the solution proposed is represented by a security protocol, which is made up of an initial handshake phase for session establishment followed by the transmission of secured messages. The former is carried out at start-up between each requester of a service and the offerer by exploiting asymmetric cryptography. Consequently, each application needs to be accompanied by a private key and the corresponding signed digital certificate, which trustworthily enumerates all the service instances (optionally through wildcards) it is allowed to either offer or request, along with the minimum security level that must be guaranteed for each of them. To this extent, digital certificates can be leveraged by car makers as a sort of contract, to certify that applications developed either internally of by third-parties are indeed allowed to request and/or offer a predetermined set of services. Fig. 2 sketches a possible format of a `vsomeip` entry within a digital certificate, containing all the required pieces of information.

To achieve complete compatibility with the SOME/IP middleware, we consider also multicast communications in addition to unicast messages. For this reason, group protection is proposed to secure the messages belonging to a specific service instance: a single symmetric key is randomly generated by the offerer and securely shared with all the requesters during the session establishment phase. Although better isolation could be provided by using multiple keys, the selected strategy is deemed to provide a good trade-off between security and complexity. Indeed, symmetric keys, being automatically regenerated every time a service is started, are assumed to last only for a limited time, reducing to a great extent the possibilities for a successful attack. Nonetheless, in case of long-running services, a re-keying mechanism would become necessary, according to a well-established practice.

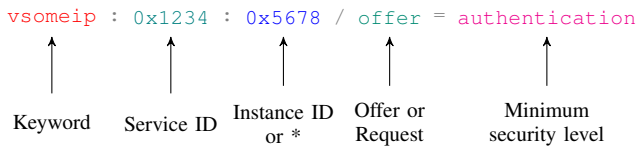

Fig. 2. Format of a `vsomeip` entry inside the digital certificate.

B. Session establishment

The purpose of the session establishment phase is twofold. On the one side, a mutual authentication is carried out, verifying that both the server and the client have respectively the right to offer and request the considered service. The former performs an explicit authentication, by means of a digital signature, while the latter is implicitly authenticated, being required to use its private key during the process. On the other one, session parameters are communicated to the requester, transmitting in an encrypted form the symmetric key necessary for the subsequent protection of messages. The actual security level of the service instance is decided by the offerer and it must be compatible with (equal to or possibly greater than) the one stated within its own certificate. At the same time, during the handshake, the requester compares the advertised security level with its own specifications, to prevent the access to a service less secure than its needs.

The handshake phase is implemented by a simple protocol, made up of two message exchanges, which follows the request/response communication pattern. Hence, all the necessary pieces of information are transported by SOME/IP packets, targeting the service for which the authentication is carried on and, in particular, a special method devoted to this task. Fig. 3 summarizes the main data exchanged during different session establishment handshakes performed in parallel by multiple applications. Every independent handshake is started by the framework on behalf of the application requesting a service, by sending an initial request to begin the communication. Most notably, the request message contains the digital certificate associated with the requesting application, trustworthily stating the list of service instances it can access. Once the offerer of the service receives the authentication request, it retrieves the peer's certificate, validates it by means of the trusted root certificate, and verifies whether the handshake can continue or the request shall be denied due to a lack of privileges.

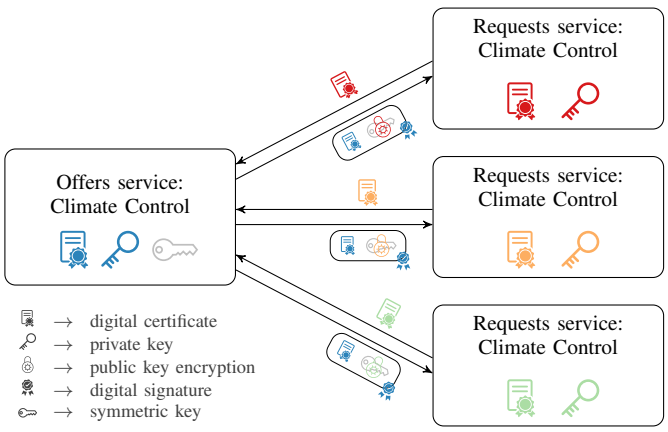

Fig. 3. Three session establishment handshakes performed in parallel by multiple applications requesting the same service, along with the exchanged messages; different colors are used to associate each element to the corresponding owner.

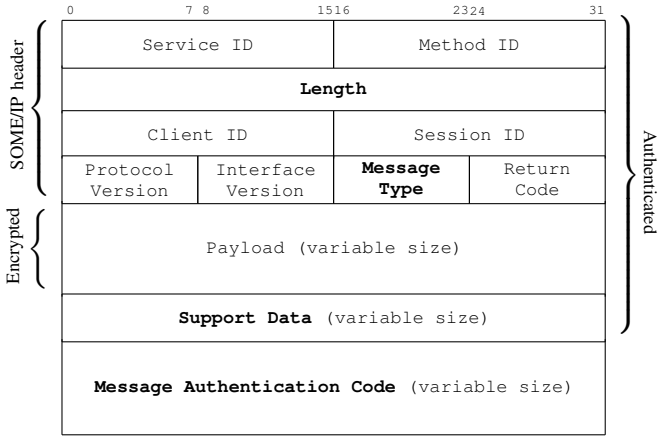

Fig. 4. Secured SOME/IP message format.

In case of a successful outcome, the response message is prepared by the offerer, to share with the requester its own certificate and the parameters necessary for the subsequent protection of the actual application messages. In particular, the response contains the symmetric key associated with the service instance of interest, encrypted with the public key stated by the certificate of the requester. This procedure guarantees that only the owner of the corresponding private key can decrypt it, thus enforcing the confidentiality of the symmetric key. Additionally, the same message comprises the digital signature computed by the offerer over the entire response to guarantee its authenticity and integrity. In the end, when the requester receives the response, it validates the certificate and verifies the permissions associated with the offerer. Finally, it can check the validity of the digital signature using the public key extracted from the digital certificate and, in case of match, decrypt the symmetric key leveraging its own private key.

The handshake protocol has been designed to be entirely executed by the communication middleware. Hence, the process is completely transparent from the applications' point of view, which are notified of the availability of the service only once the authentication has been successfully completed. Additionally, no limitations are introduced on the transport layer used by the middleware for the handshake phase. Hence, automatic retransmissions are foreseen in case of message losses and random nonces are used to associate each response to the corresponding request. Finally, although the description up to now mentioned the transmission of whole digital certificates for reasons of clarity, it would certainly be a waste of both time and network bandwidth due to their considerable size. Being the vehicle a closed system, in fact, it is possible to assume the placement of the necessary cryptographic material inside every ECU at applications deploy time. For these reasons, according to a well-established practice, certificates are actually identified during the handshake through a fingerprint, a unique identifier computed by means of a cryptographic hash function.

C. Message protection

After having successfully established a secure session, messages can be securely exchanged between the involved parties. The technique adopted for the run-time protection varies based on the security level at which the service operates. While, in case of *nosec* services, vanilla SOME/IP messages are simply serialized, *authentication* and *confidentiality*-level packets are respectively processed by the selected Message Authentication Code (MAC) [13] and Authenticated Encryption with Associated Data (AEAD) [14] algorithm. Similarly, when a message is received, its security level is immediately compared against the expected one; in addition, *authentication* and *confidentiality*-level packets are processed by the corresponding cryptographic function to verify their authenticity and, in the latter situation, to decrypt the payload: if a mismatch is detected, the message is immediately discarded.

Fig. 4 shows the format of a secured packet, highlighting in bold the differences with respect to a vanilla SOME/IP packet. While the entire message, including the SOME/IP header, is authenticated in both *authentication* and *confidentiality* levels, the latter provides also the encryption of the payload, which carries application data. The modifications are analyzed in the following:

- `Length`: since the secured packet comprises more information with respect to vanilla SOME/IP, the content of the length field needs to be updated to reflect the changes, to allow for a correct deserialization at reception side;
- `Message Type`: two previously unused bits of this field are exploited as flags, to specify the security level associated with the current message;
- `Support Data`: includes all the pieces of information required to be transmitted along with the MAC, to perform validation and decryption when the message is received; while its size and content varies depending on the adopted algorithm, it always consists of a sequence number, necessary for replay protection;
- `Message Authentication Code`: the output of the cryptographic function, which allows the receiver to verify the authenticity and integrity properties of the message; its size depends on the symmetric algorithm utilized.

The protection from message replay is guaranteed through the usage of an authenticated sequence number, added to every message of service instances operating at both *authentication* and *confidentiality* level. Being SOME/IP usable both on top of reliable and unreliable transport protocols, it is possible to assist to message losses and reordering: hence, a sliding window technique is adopted.

Finally, the proposed message format is fully compatible with vanilla SOME/IP applications; a traditional device can interact with all the existing services provided that the *nosec* security level is allowed.

IV. Experimental Evaluation

To evaluate the validity of the approach proposed, the security framework has been implemented as a PoC, integrating the designed functionalities within `vsomeip`. For the sake of simplicity, the cryptographic data has been assumed to be protected by means of operating system facilities. However, strong protection would require a hardware support, to prevent both the access to the private keys from malicious parties and the alteration of the root certificate. The source code of the PoC is publicly available on GitHub.[2]

A. Benchmark methodology

The benchmarking process concentrated on the two main phases of the proposed security protocol. First, we considered the penalties introduced by the session establishment phase, measuring the time required by an application to access the desired services, hence assessing how the solution can scale when increasing the number of services. Second, we evaluated the run-time protection phase: two applications, communicating through the `vsomeip` framework, were used to evaluate the difference between the available security levels in terms of message round trip times (RTTs), served requests per second and CPU load.

Three different strategies were adopted to achieve this goal. First, we assessed the latency introduced in the communication by the proposed security protocol, measuring the time elapsed from the very beginning of a request to the reception of the corresponding response. Hence, the output measure includes both the latency introduced by the framework and the one due to the transmission across the network. Second, we evaluated to which degree the different security measures impacted the number of requests an offerer can serve in a given unit of time. Differently from the previous evaluation scenario, the client has been configured to perform a high number of requests in parallel and at the highest possible pace, overloading both the requester and the offerer devices. In this case, the total time required to answer all the requests was used to compute the number of served requests (i.e. the corresponding response has been received by the client) per each second. Last, the alternative communication pattern, publish/subscribe, was considered, to verify whether the usage of notifications alters the results obtained with the previous techniques. As a complement of the previous benchmarks, we also evaluated the amount of CPU used by the requester while sending and receiving the messages. Although the presented measurements are referred to the client application, definitely similar results are expected to be associated with the offerer, being in charge of performing the exact same operations.

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

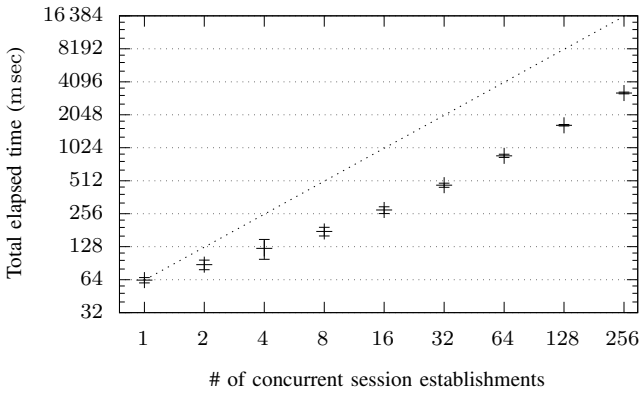

Fig. 5. Evaluation of the time required to concurrently complete multiple session establishments varying the handshake parallelism (the dotted line represents a reference corresponding to a linear increase in the elapsed time).

Being embedded systems the target of the solution presented, our testbed encompassed two identical NXP's development boards running an embedded Linux distribution and interconnected by means of a Fast Ethernet link, which represents the most common speed in the automotive environment. They are based on the i.MX 7Dual Applications Processor, characterized by two ARM Cortex-A7 cores operating at up to 1 GHz and equipped with 1 GB of DDR3 RAM. Concerning the session establishment phase, the widely used RSA-2048 asymmetric cryptosystem was chosen as a strong algorithm for encryption and digital signatures. Each RTT benchmark, on the other hand, was executed both exploiting vanilla `vsomeip`, taken as a reference, and the PoC implementation, considering all the three available security levels. For what regards *authentication* and *confidentiality*-level services, ChaCha20-Poly1305 [15] was picked up as the selected cryptographic algorithm, thanks to its outstanding performance with embedded systems. Applications based on the request/response pattern were executed considering all the three types of network bindings offered by `vsomeip`: Unix domain sockets, implementing local communication, UDP and TCP. Instead, for what regards notifications, we limited our tests to UDP since it was the only option to support multicast communication. Measurements were repeated with different payload lengths, ranging from 1 to 1024 bytes, which are deemed to be quite representative of actually used values: only the requests size was modified, while the responses were always characterized by the absence of the payload.

B. Numerical Results and Discussion

Beginning with the analysis of the session establishment benchmark, Fig. 5 plots the total time required to complete multiple authentication handshakes in parallel, when varying the amount of requested services. The overall trend certifies the scalability of the handshake phase, by showing how the measured values do not tend to explode when increasing

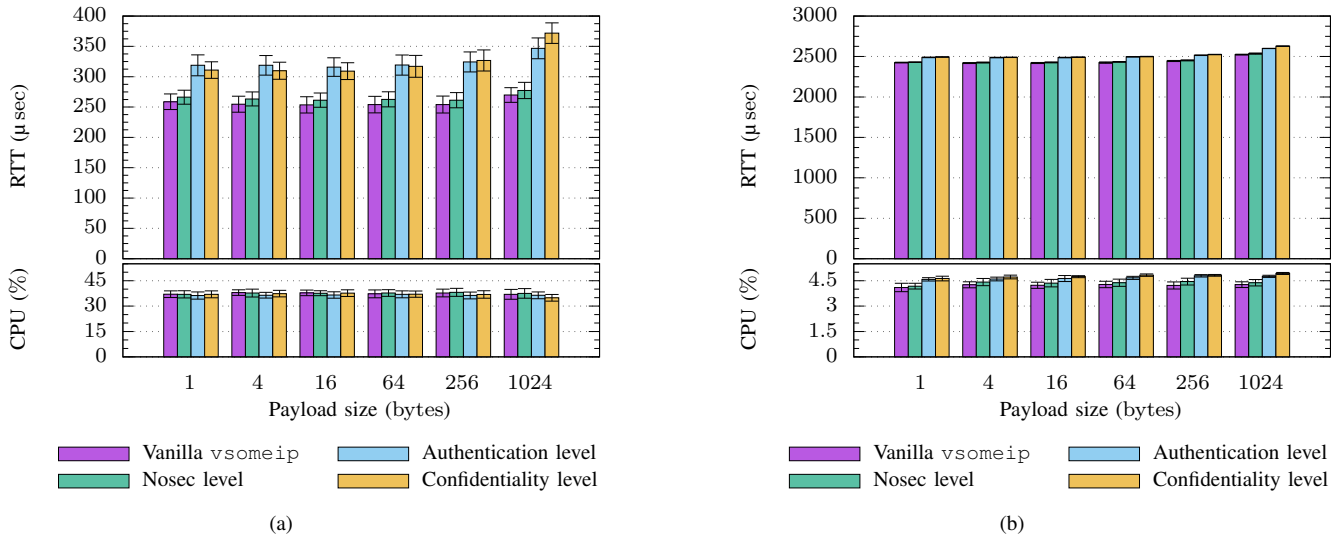

Fig. 6. RTT and CPU usage comparison between vanilla `vsomeip` and the security-enhanced version both in case of (a) local and (b) remote communication.

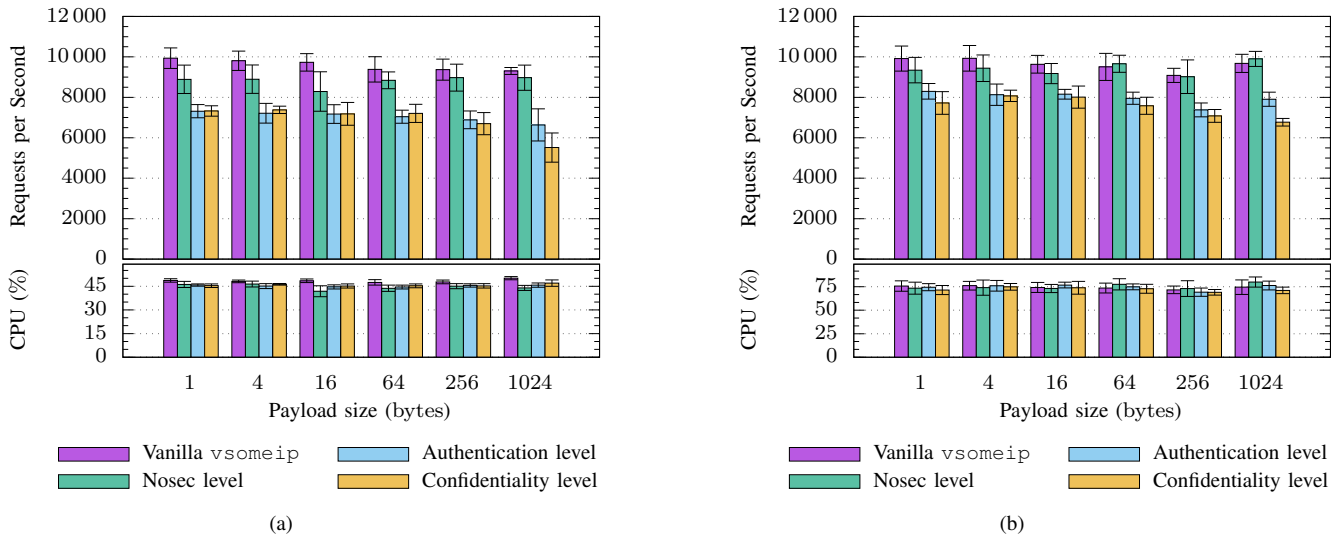

Fig. 7. Served requests per second and CPU usage comparison between vanilla `vsomeip` and the security-enhanced version both in case of (a) local and (b) remote communication.

the number of services. Indeed, considering the dotted line displayed in the graph as a reference, it becomes evident how the increase in the total time when doubling the number of concurrent session establishments is less than linear, thanks to the better exploitation of the available parallelism by interleaving the different steps of the process.

Second, the outcome of the benchmarks assessing the run-time protection is presented. Talking about transport protocols, definitely similar results have been obtained both with UDP and TCP: for the sake of brevity, only the plots about the former are presented in the following. Considering the RTT benchmarks, shown in Figs. 6a and 6b, different conclusions can be drawn depending on whether the messages need to be transmitted to a remote host or not. While in case of local communication, in fact, the security functionalities introduce additional latency accounting for about one third of the total RTT, the overhead becomes almost negligible if packets flow across Ethernet, given the predominance of the physical communication overheads. Additionally, no relevant differences are introduced by the security protocol in the measured CPU load, with at most a $10\,\%$ overhead in case of remote communication if the *confidentiality* level is selected. Nonetheless, the significantly higher number of requests issued and served when leveraging local communication (thanks to absence of the latency introduced by the physical network) imposes a considerably higher computational burden on the CPU compared to remote communication.

Figs. 7a and 7b, on the other hand, present the outcome of the benchmarks evaluating the effect of the security measures on the maximum number of requests that can be issued and served in one second. In this case, definitely similar results have been obtained both in case of local and remote communication, being the physical network overheads mitigated by the high number of parallel requests. Interestingly enough, slightly worse results

are associated with local communication: however, this behavior can be easily explained looking at the CPU load. Although both situations are characterized by rather overloaded devices, it is worth noting that, in case of local communication, both the client and the server are being executed at the same time on the same device, thus halving the available computational capacity. Hence, it becomes evident how the number of served requests is in this case limited by the CPU usage, which reached $100\,\%$. Comparing the effect of the different security levels, both *authentication* and *confidentiality* caused a decrease in the number of served requests accounting for at most one third in case of the biggest payload size considered in the evaluation, with the latter security level being slightly more demanding. Nonetheless, they introduced no relevant differences in the CPU load, being all measurements contained in the same uncertainty band.

Finally, considering the results concerning notification-based communication (not presented here due to space limitations), no evident differences can be extrapolated with respect to the simpler request/response pattern.

V. Conclusion

This paper proposed a novel approach to protect SOME/IP-based in-vehicle communications. Given the increasing cars' automation, unsecured in-vehicle messages are becoming a tempting target for wicked individuals to, e.g., conceal incomplete repairs or, in the extreme case, obtain the control of safety-critical systems. The main contribution of this paper is a novel security protocol, designed to guarantee the authenticity and confidentiality of the information exchanged without limiting the capabilities of the network middleware.

TABLE I briefly summarizes the main advantages of our proposal compared to the use of SOME/IP encapsulated within an existing secure protocol. First, both IPSec (L3 security) and TLS/DTLS (L4 security) fall short in providing all the functionalities required for full SOME/IP protection, support for one-to-many communications among all. Second, they do not fit well the communication paradigms adopted by `vsomeip`, namely the presence of a single application responsible for the transmission and reception of remote messages and the usage of inter-process communication (IPC) between applications residing on the same ECU. Indeed, they cannot provide real end-to-end security between the sending and the receiving applications, leaving the internal communications unauthenticated and protecting only the messages flowing across the network. Our solution, instead, achieves $100\,\%$ compatibility with both SOME/IP and `vsomeip`, and features a more efficient handshake procedure.

Performance measurements confirmed the introduction of limited latency ascribable to the implemented security functionalities. Considering the likely situation of UDP or TCP-based communications over a physical network, in fact, additional penalties are almost negligible, also in a constrained environment like the automotive one. Anyhow, even overloading the devices with an excessive amount of traffic, slowdowns are deemed to be still sustainable. Finally, *authentication* and *confidentiality* levels appear to be characterized by very similar performance, with a higher cost associated to the latter only for bigger payloads.

TABLE I
Comparison Between Secure SOME/IP and SOME/IP over L3 (IPSec) and L4 (TLS/DTLS) Security

	SOME/IP over IPSec	SOME/IP over (D)TLS	Secure SOME/IP
Service awareness	✗	✓	✓
Multicast support	✗	✗	✓
App-to-app security	✗	✗	✓
L4 transparency	✓	✗	✓
IPC protection	✗	✗	✓

References

[1] "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, 2016.

[2] S. Tuohy, M. Glavin, C. Hughes, E. Jones, M. Trivedi, and L. Kilmartin, "Intra-vehicle networks: A review," *IEEE Transactions on Intelligent Transportation Systems*, vol. 16, no. 2, pp. 534–545, 2015.

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

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

[5] T. Hoppe, S. Kiltz, and J. Dittmann, "Security threats to automotive CAN networks — practical examples and selected short-term countermeasures," *Reliability Engineering & System Safety*, vol. 96, no. 1, pp. 11 – 25, 2011.

[6] K. Koscher, A. Czeskis, F. Roesner, S. Patel, T. Kohno, S. Checkoway, D. McCoy, B. Kantor, D. Anderson, H. Shacham, and S. Savage, "Experimental security analysis of a modern automobile," in *Proceedings of the IEEE Symposium on Security and Privacy*, 2010, pp. 447–462.

[7] S. Checkoway, D. McCoy, B. Kantor, D. Anderson, H. Shacham, S. Savage, K. Koscher, A. Czeskis, F. Roesner, T. Kohno *et al.*, "Comprehensive experimental analyses of automotive attack surfaces." in *Proceedings of the USENIX Security Symposium*, 2011, pp. 77–92.

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

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

[10] M. Hamad, M. Nolte, and V. Prevelakis, "A framework for policy based secure intra vehicle communication," in *Proceedings of the IEEE Vehicular Networking Conference (VNC)*, 2017, pp. 1–8.

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

[12] Genivi, "vsomeip in 10 minutes." [Online]. Available: https://github.com/GENIVI/vsomeip/wiki/vsomeip-in-10-minutes

[13] C. Paar and J. Pelzl, *Message Authentication Codes (MACs)*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 319–330.

[14] P. Rogaway, "Authenticated-encryption with associated-data," in *Proceedings of the 9th ACM Conference on Computer and Communications Security*, 2002, pp. 98–107.

[15] Y. Nir and A. Langley, "ChaCha20 and Poly1305 for IETF protocols," Internet Requests for Comments, RFC Editor, RFC 8439, 2018.