CESAR: a Testbed Infrastructure to Evaluate the Efficiency of Wireless Automotive Software Updates

Marco Steger, Carlo Alberto Boano, Kay Römer, Michael Karner, Joachim Hillebrand, and Werner Rom

[‡]Virtual Vehicle Research Center, Graz, Austria

[†]Institute for Technical Informatics, Graz University of Technology, Austria

{marco.steger, michael.karner, joachim.hillebrand, werner.rom}@v2c2.at – {cboano, roemer}@tugraz.at

ABSTRACT

Connected vehicles allow to update the software (SW) running on their integrated electronic control units (ECUs) over-the-air. Such updates are complex procedures that involve several steps, such as the authentication with a remote device, the secure and reliable wireless transfer of the new binary, as well as its installation and verification on the target ECU. Each of these aspects affects the efficiency of the *entire* SW update process, and it is important to evaluate the impact of different solutions on the functionality of a vehicle and to compare their performance on real hardware. In this paper we present CESAR, a configurable testbed infrastructure that allows to evaluate the efficiency of an automotive SW update system in a highly automated way. CESAR allows to specify different update mechanisms, security configurations, wireless protocols used for the data transfer, and to carefully define the scenario of interest (i.e., pin down the number of wireless vehicle interfaces, the network topology, and the target ECU). Furthermore, CESAR can be used to measure the efficiency of a SW update on real hardware, and to derive insights about the weaknesses of a system under test or about the interaction of a specific SW with a given ECU.

KEYWORDS

Automotive Software, IEEE 802.11s, OTA Updates, Testbeds.

1 INTRODUCTION

The ability to wirelessly connect a vehicle to the Internet, to the road infrastructure, or to other vehicles, allows vehicle manufacturers (OEMs) to provide a plethora of new safety functions, comfort features, and services. Among others, automotive OEMs have the possibility to remotely diagnose a vehicle, as well as to install new SW on the ECUs over-the-air, which allows to reduce warranty costs [12]. Besides enabling performance improvements and bug fixes without the need of expensive vehicle recalls, wireless SW updates allow OEMs to upgrade or enable new features remotely.

The use of over-the-air (OTA) SW updates is not only limited to the remote download of up-to-date SW directly by the car owners (e.g., Tesla OTA updates [4]), but can also be exploited in several other stages of a vehicle's lifetime: from the vehicle development and the manufacturing stage on the assembly line, to the maintenance in a service center [13]. In all these scenarios, the vehicle uses its wireless vehicle interface (WVI) to connect to a diagnostic tester (DT) device holding the new SW binary, authorization keys, as well as other information that is required to perform the update. The update procedure itself can be conducted using *automotive diagnostic protocols* such as Unified Diagnostic Services (UDS) [1].

Due to their potential impact, OTA updates have increasingly attracted the attention of several researchers, who started analyzing the vulnerabilities of automotive eco-systems [6], and providing solutions to orchestrate secure SW updates [14]. Among others, the research community has proposed architectures to protect a vehicle from the injection of malicious SW [9, 10], and techniques to ensure reliable (wireless) inter/intra-vehicle communication [15, 17]. Most of the existing works, however, focus only on *single aspects* of an automotive SW update and not on the *entire* update process.

Need to evaluate SW updates in their entirety. The update procedure involves multiple steps ranging from the authentication with the DT and the wireless data transfer, to the installation and verification of the new binary on the target ECU. All of these aspects are interconnected and affect the overall *efficacy* and *efficiency* of a SW update, which should be always studied in its entirety. The latter requires a deep investigation of the main aspects affecting the efficiency of a SW update, such as: i) the *wireless network topology* and the number of involved nodes, ii) the applied *security configuration*, iii) the employed *SW update mechanism*, and iv) the *target ECU* and the properties of the connection to the WVI.

Need for suitable automotive testbeds. All these aspects must be evaluated in a systematic and repeatable way on real hardware (HW) to study their inter-dependency and to show the applicability of the tested SW update system. Towards this goal, it is necessary that the testbed supports not only a number of WVIs, but also their connection to one or more ECUs using automotive standard HW and SW interfaces, as well as means to install and verify the SW running on the ECU by means of diagnostic standards.

Our contributions. In this paper we present CESAR, a Configurable testbed infrastructure that allows to evaluate the effectiveness and Efficiency of wireless automotive Software updates in an Automated and Repeatable way. CESAR allows to investigate the SW update procedure in its entirety, to emulate different SW update scenarios (e.g., SW updates in a service center or in the assembly line), and to evaluate the impact of different network as well as security configurations on the update's efficiency. The proposed testbed infrastructure can be further used to analyze different ECU types, SW update techniques (e.g., the parallel or partial transfer of a firmware), and wireless communication standards (e.g., the use of single-hop or multi-hop networks).

MSWiM '17, Miami, FL, USA

978-1-4503-5162-1/17/11…$15.00

DOI: 10.1145/3127540.3127580

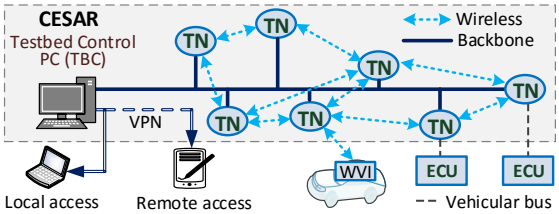

Figure 1: CESAR architecture: TNs interconnected via a backbone network allowing to connect ECUs or vehicles.

CESAR provides configuration profiles containing different node configurations (emulating different real-world scenarios), sets of parameters (e.g., key length, vehicle bus bit-rate), as well as SW update techniques. After describing its design and implementation in the next section, we show in Sect. 3 a series of case studies illustrating how CESAR can be used to evaluate the impact of different security configurations, update techniques, and network protocols on the efficiency of an automotive SW update process.

2 CESAR: DESIGN AND IMPLEMENTATION

We describe next the design and implementation of CESAR, starting from the general requirements of such a testbed infrastructure.

2.1 Testbed Requirements

A proper testbed infrastructure should support the evaluation of the *entire* automotive SW update process and allow to study the impact of different aspects on its efficiency while reducing manual intervention and allowing remote access. The employed testbed nodes must be able to support different roles (i.e., act as WVI, as DT, or as rogue node) and should be connectable to one or more ECUs from different vendors (which requires automotive HW/SW interfaces, as well as diagnostic protocols on top). Furthermore, the testbed should be able to scale up to 100 nodes while providing multiple *configuration profiles* that allow the user to choose between different network topologies and wireless communication stacks (e.g., IEEE 802.11n or 11s). These configuration profiles should include different security configurations and allow the user to choose between different security parameters, such as the authentication scheme or the key length. Ideally, also the installation effort is kept to a minimum by reusing existing network infrastructures.

2.2 Testbed Architecture

The architecture of CESAR is shown in Fig. 1: at the heart of CESAR are several testbed nodes (TNs) connected to each other wirelessly and to a testbed control PC (TBC) through a wired back-channel.

Testbed nodes. Each TN is configurable and can hence assume different roles within the testbed: it can act as a DT, WVI, relay node, or even as rogue node – a node that is compromised by an attacker. Depending on the assigned role, a TN runs a dedicated SW implementation on top of a given HW platform. The latter allows to connect a TN to an ECU using automotive bus systems (e.g., CAN or FlexRay) and easily install a new software.

ECU connection. By using automotive standards, CESAR allows to connect ECUs of different manufacturers and types, hence giving a user the ability to install SW on an ECU and to verify the success of an update procedure. In the simplest case, where different ECU SW versions periodically send CAN frames with different IDs, the verification is done on the TN acting as WVI by monitoring the CAN bus. This simple but efficient mechanism can be used for all target ECUs, even if there is no way to adapt the bootloader or flashing mechanism of the ECU. For more detailed tests, advanced features like computing the hash of the entire memory on the ECU after a SW update can be very beneficial. Therefore, we provide such features on our main target ECU, which allows CESAR to monitor the state of the ECU (via CAN) while a SW update is performed.

Configurability. CESAR provides configuration profiles allowing a simple configuration of the testbed and all its nodes. A configuration profile is a set of configuration files that can contain i) specific security and/or network configurations, ii) a certain node setting allowing to emulate specific real-world SW update scenarios, iii) a set of system parameters such as the vehicle bus bit rate or the employed authentication mechanism, and iv) specific SW update mechanisms. By utilizing the TBC, a developer can easily switch between different experimental settings and redo an experiment later by selecting this configuration profile again.

Remote monitoring. The TBC also allows other devices to access the testbed remotely. We developed a GUI that allows to monitor the state of the TNs, to set basic parameters (e.g., the wireless channel), to individually control TNs (e.g., reset TNs), and to select specific configuration profiles for planned experiments.

Prototype testing. CESAR allows to analyze the performance of different SW versions in a highly automated manner by storing and administrating all developed SW prototypes in a centralized repository. A developer can choose a specific SW version by using a specific configuration profile: prototypes are then automatically distributed to the TNs and locally configured.

SW architecture. The SW architecture of CESAR is shown in Fig. 2, and encompasses the implementations of different testbed features, the SW update system under test, and the interfaces used to interconnect the devices and the implemented prototypes. The testbed-specific SW blocks on the TBC are needed to i) (remotely) control the testbed and the running experiments, ii) retrieve specific versions of the developed DT/WVI prototypes from a GIT repository, and iii) automatically collect, pre-process and store the results of the experiments. On the TNs, testbed-specific SW blocks are required to i) assign the role of the TN, ii) locally configure the TN (e.g., vehicle bus bit-rate when connecting to an ECU), iii) enable/configure local parts of the testbed (e.g., monitoring the vehicle bus or storing debug information w.r.t. the wireless network), and iv) collect the results of an experiment and send it to the TBC.

2.3 Implementation

CESAR currently makes use of twelve TNs deployed on the ceiling of our office building, covering an area of approximately 350 m^2.

Testbed nodes. Each TN consists of a *BeagleBone Black* (BBB) board running Debian Linux and a *TL-WN722 Wi-Fi stick*, connected via USB and enabling wireless connectivity between the TNs (either IEEE 802.11n or 11s). We connect each BBB to a *custom-made PCB board* allowing the TNs to support up to two CAN connections at the same time [14]. UDS is used to perform the actual SW update procedure. Given the popularity of CAN and UDS, CESAR allows to connect a TN to almost any ECU on the market.

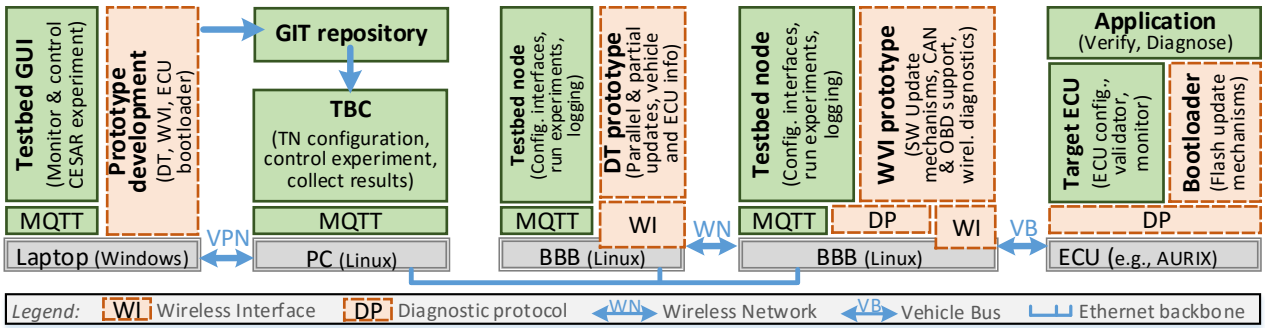

Figure 2: Software architecture. Main blocks of the testbed (green blocks, single solid line) and of the update system under test (orange blocks, dashed line). Interface are shown in blue and the employed devices using a gray block and double solid line.

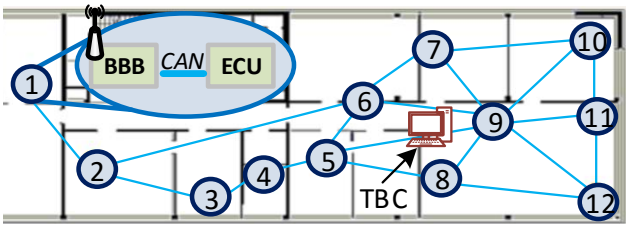

Figure 3: Position of the twelve nodes used in our testbed.

We connected the testbed nodes to two different types of ECU: the *Volvo FlexECU*, a prototype ECU used by Volvo Trucks and other automotive OEMs to test new vehicular features, and the *Infineon AURIX ECU*, a multi-core ECU used in various research and industry projects. In contrast to the Volvo FlexECU, which we had to use as black-box device without the possibility to develop our own ECU application SW, the AURIX comes with a free development tool-chain and can be powered via the I/O pins of the BBB board. This allowed us to easily connect the AURIX to the TN and to have full control on both the ECU application SW and the bootloader.

Backbone network and TBC. Each BBB board is connected to the backbone network using its Ethernet interface. To minimize the cabling effort, we exploit the existing 100 Mb/s LAN infrastructure of our office building. The TNs are decoupled from the rest of the company network infrastructure by using a dedicated subnet supporting up to 250 static IPv4 addresses. Each BBB board uses its Ethernet interface as a back-channel to communicate with the TBC. The latter is a desktop computer running Debian Linux equipped with a dual Ethernet card to connect to both the TNs and the office network infrastructure. This allows the TBC to access the Internet and other company services, such as a GIT server for source code management. To interact with the TNs, the TBC runs a Message Queue Telemetry Transport (MQTT) server. The MQTT publish-subscribe protocol allows to address all TNs at once (e.g., to start a measurement) and to configure a TN individually (e.g., assigning a specific role). Our implementation also allows to access the TBC remotely by using a VPN connection to the company network.

3 CASE STUDIES

We illustrate next a few use cases showing how CESAR can be used to study the efficiency of automotive wireless SW updates and to analyze the impact of different system configurations.

3.1 Impact of different Security Mechanisms

We first use CESAR to analyze the impact of different security mechanisms and key lengths on the duration of a SW update.

Impact of network and application layer security. We select different security configuration profiles at both application and network layer, and let CESAR automatically configure the testbed nodes with the specified security settings, while measuring the update duration and the detailed, per-step latency, using the logging ability of the DTs and the WVIs. We employ security mechanisms on the application layer implemented in SW utilizing the Java Bouncy Castle, whereas we use simultaneous authentication of equals (SAE) to protect the network layer[1]. We use the testbed deployment shown in Fig. 3 and configure node 9 to work as a DT and node 8 to act as WVI with a Volvo FlexECU connected via CAN. We choose this configuration, as it ensures a direct stable link between the DT and the WVI[2]. We perform a wireless SW update of a binary of 445 kB and measure the duration of the following steps ten times: i) Init: including WVI discovery, connection and authentication process between WVI and DT, and SW update initialization and authorization step on the ECU; ii) Upload: wireless data transfer from DT to WVI; iii) Download: data download via CAN and validation of the installed SW on the ECU.

Table 1 shows the measured overall duration and the per-step-latency w.r.t. the used security mechanisms and reveals that the data transfer from the WVI to the ECU via CAN takes the largest portion: about 75% of the overall duration with all security features enabled, and up to 87% if these mechanisms are disabled. The results also expose that the security functions have a significant impact on the update duration: plus 18.5% when all mechanisms are enabled.

Impact of the key length on the update duration. We use the aforementioned experimental setup and configure CESAR to disable the security mechanisms on the network layer. Different configuration profiles are then used to evaluate the impact of the key length on the update duration. Specifically, we use different key lengths for both the RSA-based authentication and AES-based data encryption, and perform 10 sequential wireless SW updates using the AURIX ECU for each configuration. Table 2 shows the duration of a SW update depending on the employed key length: varying the key length of the RSA-based authentication has a stronger impact on the update duration than different AES encryption key lengths.

[1]For more details on the utilized security mechanisms, we refer the reader to [13].
[2]CESAR would also allow a more difficult setup with links of intermediate quality.

Table 1: Duration in ms of the different wireless SW update steps depending on the employed security mechanism.

Security	Total	Init&Auth.	Upload	Download
Appl. + Net.	49195.4	4782.3	6585.4	37817.7
Application	47167.1	4859.7	5414.2	36885.3
Network	43745.6	2277.7	3756.0	37703.3
None	41528.6	2277.9	2445.0	36797.5

Table 2: Impact of the key length on the update duration

RSA	AES	Duration	Delta
1024	128	16271.0 ± 323.4 ms	-
1024	256	16375.1 ± 222.3 ms	104.1 ms (+0,6%)
2048	128	18342.7 ± 357.4 ms	2071.7 ms (+12.7%)
2048	256	18359.1 ± 292.3 ms	2088.1 ms (+12.8%)

Table 3: Update duration w.r.t different update mechanisms.

Traditional	Parallel	Partial
20768.8 ± 882.4 ms	25881.8 ± 324.5 ms	3570.7 ± 1189.3 ms

3.2 Impact of different ECU Hardware

We illustrate CESAR's ability to support ECUs of different vendors by connecting a WVI device to both a Volvo FlexECU and an AURIX ECU via CAN (at 500 kb/s). We employ node 9 as DT and node 8 as WVI according to the testbed deployment shown in Fig. 3. As the Volvo FlexECU can only be used as a black-box, we create a dummy application for the AURIX that has the same binary size as the one available for the Volvo FlexECU (i.e., a size of 445 kB). We then run twenty wireless SW updates on each of the two ECUs, and let CESAR measure their average duration a discussed earlier.

The gathered results show that the AURIX ECU can be updated about three times faster than the Volvo FlexECU: the SW update takes indeed 20.77±0.88 and 48.68±0.81 seconds on the AURIX and the FlexECU, respectively. This is due to (i) the higher CPU power and faster storage modules of the AURIX ECU, and (ii) the fact that the FlexECU uses a two-stage update procedure encompassing a secondary bootloader and the application SW itself.

3.3 Impact of different Update Techniques

CESAR can also be used to evaluate the efficiency of different SW update mechanisms: traditional, parallel, and partial updates.

Parallel updates. We first compare the duration of the update process when using traditional SW updates sequentially with the duration of a *parallel* SW update. A parallel SW update is performed on two or more ECUs integrated in two or more vehicles at the same time. For an experiment within CESAR this means that new SW is installed on several (in this particular experiment two) ECUs at the same time by one DT (node 9). Therefore, each ECU is connected to a testbed node acting as WVI (nodes 8 and 11) and the SW update is first done sequentially (first node 8 is updated and then node 11) and then in parallel, meaning on both ECUs at the same time.

Table 3 shows that the overall duration of one parallel SW update (for two ECUs) is increased by about 25% compared to a traditional wireless SW update (for one of the ECUs). This overhead is due to the fact that not all steps of a SW update can be done in parallel. However, carrying out parallel SW updates is significantly faster (approximately 75% quicker) compared to a sequential updates.

Table 4: Update duration w.r.t. different network protocols.

802.11s duration	802.11n duration	TN	RSSI
16590.4 ± 286.1 ms	16545.0 ± 360.8 ms	6	-83 dBm
16943.0 ± 453.1 ms	16504.7 ± 238.4 ms	4	-87 dBm
16918.2 ± 518.4 ms	21274.8 ± 4632.9 ms	2	-90 dBm
17620.1 ± 1696.6 ms	Unreachable	1	>90

Partial updates. A SW update is often only changing specific parameters of an ECU, leaving most of the remaining SW untouched. For this reason, it may be advisable to only update the changed SW portion, instead of the entire binary. We compare next the duration of a traditional SW update with a custom implementation of a *partial* update in which only the portion of code that has changed is installed. We prepare two SW applications with a parameter field stored in a dedicated memory section of the AURIX ECU of size 1 kB: this parameter field is the only difference between the binaries. When utilizing a partial SW update, only this section is transferred to the ECU. Instead, when using a traditional SW update, the entire binary (of size 445 kB) needs to be transferred. The case study was performed using node 9 as DT and node 8 as WVI connected to an AURIX ECU. We performed twenty SW updates using both the traditional and the partial SW update mode.

Table 3 highlights that, as expected, the SW update duration is significantly reduced: the partial update is about six times faster. This decrease in duration of 83% is especially due to the lower amount of data transferred from the WVI to the ECU via CAN.

3.4 Impact of different Network Protocols

In this case study, we evaluate the efficiency of SW updates of two different wireless protocols: IEEE 802.11n and IEEE 802.11s. The key difference between these two protocols is the ability to build mesh networks: IEEE 802.11n is the traditional, access point based Wi-Fi protocol. Instead, IEEE 802.11s allows to form multi-hop networks increasing the reliability and availability of the entire network.

In our experimental setup illustrated in Fig. 3, we select node 11 as DT and different TNs (nodes 1, 2, 4, and 6) connected to an AURIX ECU. For the IEEE 802.11n evaluation, the DT node is also acting as wireless access point. It is important to highlight that, in this configuration, the links between 11 and the other TNs are of different quality. Furthermore, node 1 is the furthest away, and is not in the communication range of node 11. As a result, when employing IEEE 802.11n, no communication can be established between the two nodes. In contrast, when using IEEE 802.11s, the nodes can decide to hop through additional nodes to use only very good links and maximize the reliability of communications.

We then perform ten SW updates for each configuration and let CESAR measure the SW update duration (Table 4). For good links (node 4 and 6), both protocols nearly exhibit the same performance. For node 4, IEEE 802.11n, on average, is about 400 ms faster than IEEE 802.11s. The experimental results show an average path length of 1.06 hops when using IEEE 802.11s. This means that IEEE 802.11s has employed some multi-hop paths during the SW update process (due to lost packets) leading to a slightly increased packet latency. In case of intermediate links (node 2), IEEE 802.11s outperforms IEEE 802.11n by a factor of 25%.

3.5 Connectivity issues of IEEE 802.11s

CESAR can also be used to investigate the connectivity of wireless nodes and thereby, as presented in this case study, to reveal scalability issues. When configuring CESAR to use IEEE 802.11s, we observed that some TNs were not reachable by other nodes, despite being in close proximity. The observed issue is critical as typical SW update scenarios can encompass several vehicles in a dynamic environment frequently joining and leaving the network. To analyze the problem, snapshots of the connectivity in the network (i.e., link and path information) were captured using CESAR. These tests reveal two major problems of the default open11s implementation caused by its limited neighbor table size:

Inefficient network structure. In IEEE 802.11s, peer links are established between two nodes if i) the nodes can *hear* each other and ii) the nodes have a free spot in their neighbor table. Hence, the network topology of 11s is mainly influenced by the sequence of nodes joining the network (and not if a node is close or far away) and thus leads to inefficient links affecting the network performance.

Isolated node. In the worst case the limited neighbor table size (e.g., size=3) can even lead to isolated nodes: a node (e.g., node 5) willing to join an already established network, will fail to connect as the other nodes (e.g., nodes 1 to 4) already have three neighbors stored in their neighbor table. The nodes will decline the join requests and node 5 will be isolated from the network.

We solved this issue by adapting the latest `open11s` implementation: i) adding new messages to inform the network about isolated nodes, and ii) implementing algorithms to solve the *isolated node problem*. The adapted `open11s` version can be chosen and configured by CESAR besides the default `open11s` version.

4 RELATED WORK

In this section, we summarize the body of existing works focusing on wireless automotive SW updates and compare the functionality of existing automotive testbeds to the ones offered by CESAR.

Wireless automotive SW updates. Most of the work on automotive OTA updates has focused on security aspects and proposed security architectures protecting a vehicle from malicious updates [9, 11] or investigated remote SW updates [7, 10]. These works, however, do not consider other scenarios where updates are performed locally (such as within a service center), nor allow testing of advanced update mechanisms such as the parallel transfer of a binary. Solutions are also often evaluated only through simulation [5, 11] or formal methods [9, 10, 13], and very few systems are evaluated on real HW. In this and our previous works [8, 14], automotive ECUs are used to evaluate a SW update framework, verify the update process, and compare different update mechanisms.

Automotive testbeds. Several infrastructures have been proposed to test automotive SW updates. Drolia et al. [3] have designed a testbed consisting of several automotive ECUs interconnected by CAN. Although the testbed provides a basic SW update function, it is not capable of evaluating the entire wireless SW update process. In [16] and [2], authors have proposed Vehicle-to-Vehicle testbeds (either indoor [16] or outdoor [2]) to simulate different V2V scenarios. However, these testbed do not support automotive ECUs and cannot be used to evaluate any aspect w.r.t wireless SW updates.

5 CONCLUSIONS

In this paper we propose CESAR, a testbed infrastructure that allows to investigate the efficiency and dependability of an entire wireless automotive SW update process. After describing the testbed architecture and design, we show through a series of case studies that CESAR allows to derive insights about the impact of different security configurations, update techniques, and network protocols on the efficiency of an automotive SW update process. In the future, we plan to use CESAR to evaluate the reliability of IEEE 802.11s and to run different attacks on the SW update framework presented in [13], in order to evaluate its robustness and expose its weaknesses.

Acknowledgments. This work was partially funded by the SCOTT project. SCOTT (http://www.scott-project.eu) has received funding from the Electronic Component Systems for European Leadership Joint Undertaking under grant agreement No 737422. This joint undertaking receives support from the European Unions Horizon 2020 research and innovation programme and Austria, Spain, Finland, Ireland, Sweden, Germany, Poland, Portugal, Netherlands, Belgium, Norway. SCOTT is also funded by the Austrian Federal Ministry of Transport, Innovation and Technology (BMVIT) under the program "ICT of the Future" between May 2017 and April 2020. More information at https://iktderzukunft.at/en/. The authors also acknowledge the financial support of the COMET K2 Program of the Austrian Federal Ministries BMVIT and BMWFW, the Province of Styria, and the Styrian Business Promotion Agency (SFG).

REFERENCES

[1] ISO 14229:2006(E). 2006. ISO 14229:2006: Road vehicles – Unified diagnostic services (UDS) – Specification and requirements. (2006).
[2] M. Cesana, L. Fratta, M. Gerla, E. Giordano, and G. Pau. 2010. C-VeT the UCLA Campus Vehicular Testbed: Integration of VANET and Mesh Networks. In *Proc. of the European Wireless Conference.*
[3] U. Drolia, Z. Wang, S. Vemuri, M. Behl, and R. Mangharam. 2011. AutoPlug – An automotive test-bed for Electronic Controller Unit Testing and Verification. In *Proc. of the IEEE Intelligent Transportation Systems Conference (ITSC).*
[4] N. Gabe. 2016. Over-the-Air Updates on Varied Paths. *Automotive News* (2016).
[5] I. Hossain, S.M. Mahmud, and M.H. Hwang. 2010. Performance Evaluation of Mobile Multicast Session Initialization Techniques for Remote SW Upload in Vehicle ECUs. In *Proc. of the IEEE Vehicular Technology Conference.*
[6] K. Koscher et al. 2010. Experimental Security Analysis of a Modern Automobile. In *Proc. of the IEEE Symposium on Security and Privacy.*
[7] M. Khurram et al. 2016. Enhancing Connected Car Adoption: Security and OTA Update Framework. In *Proc. of the 3rd World Forum on Internet of Things.*
[8] M. Steger et al. 2017. An Efficient and Secure Automotive Wireless Software Update Framework. *Under submission* (2017).
[9] M.S. Idrees et al. 2011. Secure Automotive On-board Protocols: A Case of Over-the-air Firmware Updates. (2011).
[10] D.K. Nilsson and U.E. Larson. 2008. Secure Firmware Updates Over the Air in Intelligent Vehicles. *IEEE Conference on Communications* (2008).
[11] R. Petri et al. 2016. Evaluation of Lightweight TPMs for Automotive SW Updates over the Air. In *Proc. of the Conference on Embedded Security in Cars.*
[12] Redbend Software. 2011. Updating Car ECUs Over-The-Air (FOTA). (2011).
[13] M. Steger, C.A. Boano, M. Karner, J. Hillebrand, W. Rom, and K. Roemer. 2016. SecUp: Secure and Efficient Wireless Software Updates for Vehicles. In *Proc. of the Conference on Digital System Design (DSD).*
[14] M. Steger, M. Karner, J. Hillebrand, W. Rom, C.A. Boano, and K. Roemer. 2016. Generic Framework Enabling Secure and Efficient Automotive Wireless SW Updates. In *Proc. of the Conf. on Emerging Technologies and Factory Automation.*
[15] I. Studnia, V. Nicomette, E. Alata, Y. Deswarte, M. Kaâniche, and Y. Laarouchi. 2013. Survey on Security Threats and Protection Mechanisms in Embedded Automotive Networks. In *Proc. of the Conf. on Dependable Systems and Networks.*
[16] W. Vandenberghe, I. Moerman, P. Demeester, and H. Cappelle. 2011. Suitability of the wireless testbed w-iLab.t for VANET research. In *Proc. of the 18th Symposium on Communications and Vehicular Technology in the Benelux.*
[17] T.L. Willke, P. Tientrakool, and N.F. Maxemchuk. 2009. A Survey of Inter-Vehicle Communication Protocols and their Applications. *IEEE Communications Surveys & Tutorials* 11, 2 (2009).