Security for low-end automotive sensors: a tire-pressure and rain-light sensors case study

Adrian Musuroi[1], Bogdan Groza[1], Stefan Murvay[1] and Horatiu Gurban[1]
[1]*Faculty of Automatics and Computers, Politehnica University, Vasile Parvan Boulevard, Timisoara, Romania*
adi.musuroi95@gmail.com, {bogdan.groza, pal-stefan.murvay, eugen.gurban}@aut.upt.ro

Keywords: sensor, tire pressure monitoring, rain-light sensors

Abstract: In recent years, the security of in-vehicle buses and components has been extensively addressed, but only a few research works considered the security of low-end in-vehicle sensors. The main problem in addressing such components stems from the numerous constraints, both in terms of computational power, since most sensors are equipped with low-speed 8 bit controllers, and low bandwidth. In this work we use as a case study a tire-pressure sensor and a rain-light sensor. The first communicates over radio-frequency while the second uses a low-speed in-vehicle bus, both interfaces having a very low bandwidth and reduced packet size of only 64 bits. Under these constraints we discuss the design of a cryptographic security protocol based on an existing lightweight block cipher in order to assure both security and privacy objectives.

1 Introduction and motivation

In this work we address security and privacy issues related to tire-pressure and rain-light sensors. The security of vehicle subsystems has been carefully examined in the recent years, e.g., (Koscher et al., 2010), (Checkoway et al., 2011), (Miller and Valasek, 2014). Also, security issues regarding sensors in autonomous vehicles have been pointed out by authors in (Yan et al., 2016). But due to a somewhat reduced impact on safety and security, there are not many works that focused on the security of low-end components such as tire pressure sensors or rain-light sensors. Still, the insecurity of these components may cause serious concerns as these components are present on many recently manufactured vehicles. In particular, tire pressure monitoring systems (TPMS) are currently mandatory in order to increase traffic safety and reduce pollution (van Zyl et al., 2013), (Ergen and Sangiovanni-Vincentelli, 2017). What these sensors have in common is that they have communication interfaces with very low bandwidth and a reduced packet size of only 64 bits. The first communicates over radio-frequency while the second is wired and uses a low-speed in-vehicle bus, the Local Interconnect Network (LIN).

For tire pressure monitoring systems (TPMS) both security and privacy concerns have been previously raised (Ishtiaq Roufa et al., 2010). In particular, tracing vehicles based on the unique sensors IDs may be a serious concern regarding the privacy of users. We discuss some related work on TPMS security in the next section. For rain-light sensors (RLS) there is no research so far, but due to the safety-critical features to which they are linked, i.e., automatic control of lights or windshield wipers which may affect road visibility, addressing their security seems also to be relevant. Since recent research works have pointed out security issues of the LIN bus (Takahashi et al., 2017) to which these sensors are connected, future attacks are possible.

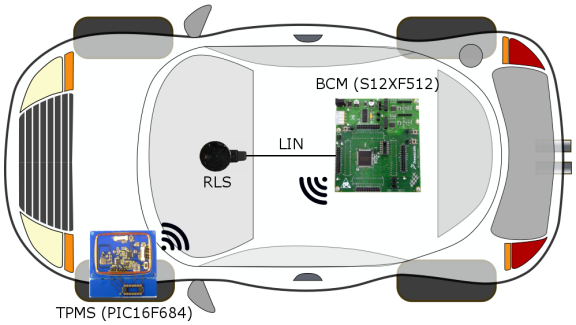

Figure 1: Addressed scenario

Our paper is organized as follows. In the next subsection we discuss some related work on TPMS and background on RLS. Section 2 presents results on eavesdropping data from real-world TPMS sensors and gives a clearer image on the addressed problem. Then in section 3 we present the experimental setup,

discuss our protocol design and present some experimental results. Finally, section 4 holds the conclusions of our work.

1.1 Existing solutions

The automotive industry introduces new comfort and safety features with each new generation of vehicles. Both tire-pressure monitoring systems (TPMS) and rain-light sensors (RLS) are good examples in this direction.

Following the attacks proposed in (Ishtiaq Roufa et al., 2010) on TPMS several solutions were proposed in the literature. One of the first solutions was discussed by authors in (Xu et al., 2013) but the experimental results are on the Arduino platform which is not a real-world TPM platform. A patent from Continental Corporation exists in (Toth, 2014) but the solution does not appear to be secure since it is based on CRC codes which are not a cryptographically secure building block. A low-cost solution on a real-world TPM sensor, i.e., the Infineon SP37 platform, was developed by the authors in (Solomon and Groza, 2015) and similar to ours uses the SPECK 32/64 block cipher (Beaulieu et al., 2015). In contrast to this, here we use a distinct implementation platform, we focus more on limitations of existing TPMS solutions and go toward a more compact packet with SPECK 48/96 that saves some computational time (since it allows to embed the entire data-field in a single encryption block).

The RLS is a sensor able to detect rain intensity and the ambient brightness. By detecting the raindrops on the windshield it can provide the information needed to control the activation/deactivation and to adjust the speed of the windshield wipers. The ambient brightness is used to control whether the daytime running lights or the low beam lights are switched on. The RLS is mounted on the interior upper part of the windshield, above the rear-view mirror.

The information provided by the RLS are also used by some automotive manufacturers to close the windows/sunroof in case of rain, to modify the light intensity of different interface elements: instrument cluster, head-up display, infotainment unit display, and other illuminated switches and buttons. There are also variations of the RLS sensor with an integrated humidity sensor that allows automatic windshield ventilation for preventing windshield fogging.

We focused our attention on a 4th generation RLS sensor from Hella. The RLS is usually connected to a BCM module by using the Local Interconnect Network (LIN) protocol. As shown in Figure 2, which illustrates the structure of a LIN frame, the LIN data field is limited to carrying 64 bits of data. The RLS LIN frame used by our target sensor device can be seen in Figure 3 (the PID field is blurred due to confidentiality reasons). The RLS that we studied uses a Renesas μPD78F1817 16-bit low-power microcontroller with a maximum operating speed of 24MHz. Newer RLS devices from the same family use Renesas R5F10AGG chips. Characteristics for both microcontrollers can be found in Table 1. To the best of our knowledge there are no attacks reported so far on such sensors and no countermeasures discussed in the literature.

The RLS being located on the windshield, the LIN bus wires have to cross a part of the car ceiling and go trough one of the car pillars to connect in the Body Control Module (BCM). The BCM acts as the LIN bus master in this case. There aren't many other in-vehicle devices that may connect to the ceiling LIN wire which makes the use of RF communication for the RLS a plausible option for reducing wiring. RF communication is a viable alternative considering that the LIN bus is characterized by low-speed communication requirements and a reduced frame length. The RLS does not have to be powered by its own battery like TPMS sensors since power cables are already available on the car ceiling for assuring ambient light. Our after-market RLS sensor is not programmable, consequently, in the experimental section we focus on the TPMS unit. But since from a computational perspective the RLS sensor is superior to the TPMS sensor that we used, the proposed solution can be easily deployed in any of the systems.

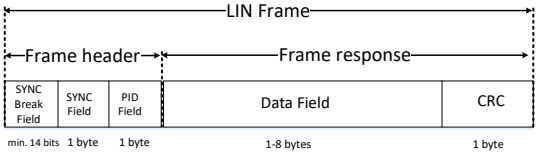

Figure 2: Structure of a LIN frame

2 Assessing the problem on real-world TPMS sensors

Before proceeding to the design of a security protocol, we conducted experiments on real-world sensors in order to determine the feasibility of practical attacks. This would give a crisper image on the security problems related to TPMS. This section discusses about eavesdropping sensor data from existing in-vehicle sensors.

Exploiting TPMS vulnerability to identify and

Table 1: Characteristics of RLS microcontrolers and components of our experimental setup

Function	Microcontroller	Architecture	Operating frequency	RAM	Flash	EEPROM	Connectivity
RLS	µPD78F1817	16-bit	24MHz	8KB	128KB	16KB	CSI with SPI support, simplified I2C, LIN/UART
RLS	R5F10AGG	16-bit	32MHz	8KB	128KB	4KB	CSI, simplified I2C,LIN/UART
BCM	S12XF512	16-bit	100MHz	32KB	512KB	4KB	CAN, FlexRay, SPI, SCI with LIN support
TPMS	PIC16F684	8-bit	8MHz	128B	2Kword	256B	RF

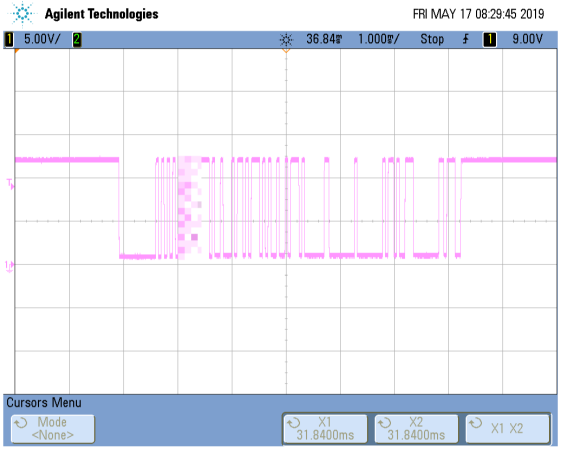

Figure 3: RLS LIN frame

track vehicles requires that the adversary knows at least one sensor ID for each target car. Finding these values is not straightforward, because the data packet format is not standard for all cars, thus the length and positioning of the ID within the frame varies. Research work done in (Ishtiaq Roufa et al., 2010) shows that each TPMS sensor transmits a 28 - 32 bit ID, which makes a car with 4 such sensors identifiable and thus traceable. In (Ishtiaq Roufa et al., 2010), the method used for finding the packet format was by observing the changes in the data frame after changing only one measured variable at a time (tire temperature or pressure). This reveals the position of the corresponding variable in the frame. The bytes that never change their values are suited candidates for the ID. Normal pressure and temperature fluctuations due to the wheel friction are sufficient to determine the packet fields only if a large number of captured frames is available. Otherwise, more drastic changes are required and it may not be possible for an adversary in a real context to increase the tire temperature using a heat gun or to change the pressure by inflation or deflation.

Tools for reverse engineering TPMS sensor data are already available on-line. Free software is available for statistically inspecting TPMS frames that were recorded from a moving vehicle. [1] However, there are some considerations to be taken into account. First, this method requires time and expertise because the software cannot decode the messages by itself. Information such as the symbol rate and preamble format must be provided by the user, considerably reducing the adversary space. Secondly, because the software is based on statistical analysis, we must assume that the adversary is capable of recording a relatively high number of messages with as few interceptions as possible (i.e. transmissions from other vehicles). The requirements for this task is that the adversary is able to stay in close vicinity of the moving vehicle (TPMS messages are usually transmitted only when a certain speed is reached by the vehicle) while making sure that there are as few other vehicles as possible around so there are not many interceptions. Considering also that the transmission rate is approximately one minute, gathering a sufficient amount of captures may be somewhat inconvenient for practical reasons.

In our approach, we conducted TPMS reverse engineering by using a commercial kit of programmable sensors that are compatible with 98% of the vehicles on the market. When programming the sensors, the associated hardware and software tools allow us to select a particular vehicle (i.e. brand and model) and the sensor ID, the latter having the options to be generated within the software or handwritten. Automatic ID generation indicates the specific ID length for each vehicle model. One observation is that continuously generating IDs will not change the value of the most significant byte until other vehicle model is selected. This indicates a possible common practice of using IDs with the same most significant byte value for a given vehicle. Further, the sensors can be diagnosed. During this process, the kit emulates a BCM that triggers the sensor to send data via RF. Searching the previously programmed ID in sensor transmissions allows us to precisely determine its position in the data frame.

The RF signal was captured using a HackRF One receiver [2] with GNU Radio Companion and further inspection, demodulation, symbol rate recovery and symbol extraction were all made using Inspectrum. [3] Figure 4 shows the HackRF tool and a commercial tire sensor. In Figure 5 we show a TPMS trace captured from the sensor during diagnosing. The cursors

[1] https://github.com/jboone/tpms

[2] https://greatscottgadgets.com/hackrf/one/

[3] https://github.com/miek/inspectrum

Figure 4: Analysis tools: HackRF and a commercial tire sensor

are manually placed to cover the entire signal, giving a raw signal length of 176 symbols. More useful information such as the symbol rate and symbol period are displayed in the Controls view. After extracting the symbols, removing the preamble and applying Manchester decoding (most common encoding technique used), the previously programmed three-byte ID is revealed within the data frame.

Further, we applied the same process for sensors programmed for several vehicle brands and models chosen from the Jato Dynamics 2018 best selling car statistics, [4] but also for other popular brands not included in this chart. We find that there are various patterns for TPMS messages and we used them to group vehicle models. Table 2 summarizes our results for eight identified patterns. For each one, the preamble, modulation method, encoding method, symbol rate, ID length and ID positioning are shown. Also, in the last column, we show the order in which the ID bytes are sent, i.e. most or least significant byte first. One mention is that for preamble identification we selected the maximum number of leading symbols that include repeated sequences of '10'/'01', or sequences that cannot be a result of Manchester encoding (e.g. a sequence of four 1's). Values from the ID offset column were determined after removing the preamble.

From patterns no. 2, 4 and 5 we learn that multiple encoding techniques may be used in TPMS RF transmissions. In our case, Non Return to Zero Inverted encoding is used along Manchester, the latter being the only encoding method considered in other research works. Using multiple encodings adds obscurity thus making reverse engineering more complex, however it is unknown if this is done as a security measure. Pattern no. 1 sends the ID starting with the least significant byte first while the others start with the most significant one. The transmission order is relevant in a scenario in which an adversary attempts to track a vehicle with known IDs.

To validate our method of TPMS reverse engineering, we attempted to recover the sensor IDs of a 2017 vehicle equipped with stock factory sensors. Due to privacy concerns, the vehicle brand or model are not disclosed. Driving the vehicle while recording TPMS's specific frequency resulted in a rich set of captured frames from our vehicle but also from others. While inspecting the captures, we found that almost all of them match one of our previously found patterns based on raw length, preamble and symbol rate. This is very promising considering that our list of patterns is not exhaustive and that there are other existing devices using the same transmission frequency. Further, we selected 70 messages matching our specific vehicle's group pattern and extracted the values from the supposed ID position in the frame. As expected, we found only four different values, which are the following: 0x28ccfe56, 0x28cd094b, 0x28ccf426 and 0x28cdce57. Note that the most significant byte has the same value for all four IDs, consistent with our earlier observation.

Attack automation. By using a TPMS kit similar to ours, one can create a database containing information such as baud rate, transmission length, message format, etc. for TPMS sensors of most modern vehicles. Attacks such as the one described by us can be automated and specialized tools may be developed. This method would allow adversaries with no expertise and with only one capture from each sensor to retrieve all four IDs of a vehicle, raising a serious concern about the security of current TPMS implementations.

Active attacks. Spoofing attacks were also discussed in (Ishtiaq Roufa et al., 2010). Authors showed that injecting forged messages containing critical pressure data can mislead the tire pressure monitoring system and trigger warning messages to the driver. Having at least one sensor ID of a vehicle, one adversary can program the value to a commercial TPMS sensor, which will generate valid messages, but with data measured from an environment controlled by the adversary. By eavesdropping TPMS messages we found no evidence of any freshness mechanisms being used, thus messages can be recorded and later replayed to a targeted vehicle. The advantage of this method is obvious. No software-defined radio or error checking techniques (i.e. CRC) knowledge is required when computing a forged message.

[4] https://www.best-selling-cars.com/europe/2018-full-year-europe-best-selling-car-models-in-the-eu/

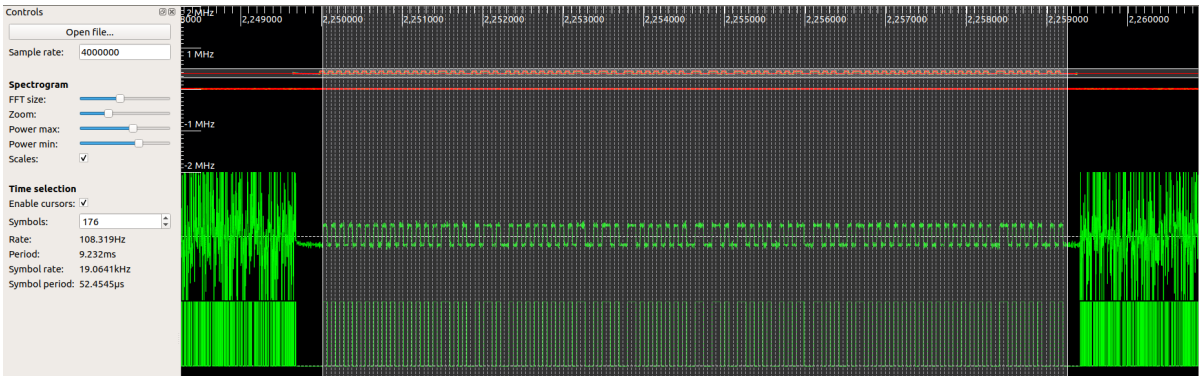

Figure 5: Captured TPMS message viewed in Inspectrum

Table 2: Interpreting extracted data from sensors

Pattern no.	Total symbols	Preamble	Modulation	Encoding	Symbol rate	ID length (bytes)	ID offset (bits)	MSBF/ LSBF
1	176	15555555h	FSK	Manchester	19 kHz	3	25	LSBF
2	208	AAAAAAh	FSK	Manchester	38 kHz	4	12	MSBF
2	220	2Ah	FSK	Manchester, NRZI	19 kHz	4	18	MSBF
3	159	15555555h	FSK	Manchester	19 kHz	4	1	MSBF
4	180	39999999999981Fh	FSK	Manchester, NRZI	19 kHz	3.5	2	MSBF
5	159	2A9Eh	FSK	Manchester, NRZI	19 kHz	3.5	4	MSBF
6	138	3D55h	ASK	Manchester	8 kHz	3.5	10	MSBF
7	193	15555555h	FSK	Manchester	19 kHz	4	9	MSBF
8	223	15555555h	FSK	Manchester	19 kHz	4	1	MSBF

3 Setup and proposed protocol

This section discuses the platforms that supported our protocol implementation as well as details on our protocol design. Finally we discuss performance results in terms of computational power an power consumption.

3.1 Embedded platforms

Two embedded platforms were employed as support for our experiments. A TPMS kit from Microchip (shown in Figure 6) based on PIC16F684 was chosen as the low-end automotive TPMS sensor. The kit comes with a base-station that receives information from the sensor. However, the base station is not an automotive-grade controller, a reason for which we also included in our experiments an NXP S12XF based EVB9S12XF512E development board. The S12 family is commonly used in the automotive industry and thus having results on this platform is at least worthy as a comparison. The RLS sensor that we have is an after-market component and is not programmable. In terms of computational power, it lies between our PIC controller and the S12 board, thus the experimental results on the two should be convincing regarding the RLS as well.

PIC16F684 based sensor. Our TPMS sensor development platform from Microchip is built on a 8-bit PIC16F684 microcontroller with an operation frequency of up to 8 MHz. In terms of memory, the chip features 2 Kwords of flash (program memory is expressed in words with 1 word = 14 bits), 128 Bytes of RAM and 256 Bytes of EEPROM. From the operational point of view, the sensor is triggered by 125 kHz low-frequency (LF) messages sent by the base station trough a LF Initiator module. When a valid LF message is received by the on-board three-channel Analog Front End (MCP2030), an interrupt is generated to wake up the processor from low-power mode. Pressure, temperature and battery level are measured and the data is sent back to the base station via 433.92 MHz radio frequency (RF). The actual sensor used by Microchip in the sensor module design is an MS5407-AM from Intersema, which is suited for this type of applications.

S12XF based receiver. Most functionalities related to the vehicle body functional domain are usually implemented by a single unit called BCM. Since TPM and RLS fall into this category we opted for a receiver platform that is recommended for vehicle body applications. The NXP S12X microcontroller family is an option recommended for such applications. We employed the EVB9S12XF512E development board, equipped with an S12XF512 microcontroller, as the BCM. The S12XF512 microcontroller provides 32 KBytes of RAM, 512 KBytes of Flash and a 16-bit main core that can operate at frequencies up to 100 MHz. Its communication capabilities cover options for interacting with other in-vehicle modules via protocols such as CAN or FlexRay as well as protocols to interface with various sensors and periph-

erals, i.e. LIN, SPI and asynchronous serial communication. This makes it easy to connect the microcontroller to on-board sensors and radio-frequency transceivers enabling monitoring abilities for a wide range of sensors.

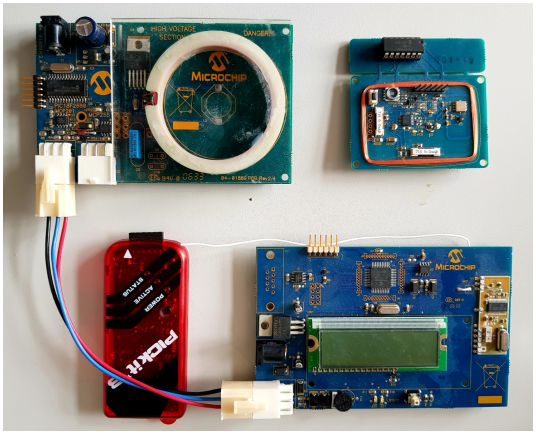

Figure 6: Microchip TPMS kit

3.2 Protocol design goals

As showed in section 2 and by related work, TPMS sensors send data packets in plain, revealing IDs sufficiently large for an adversary to track vehicles. Also, the lack of freshness and authentication allows spoofing and replay attacks to be performed undetected. Our protocol aims to improve TPMS security, while keeping the added time, memory and energy costs as low as possible. Current standards in automotive on-board communication request for authentication tags of 24 bits and freshness parameters (AUTOSAR, 2017). Specifications for interfaces of cryptographic primitives also exist in (AUTOSAR, 2015a) and (AUTOSAR, 2015b). For authenticity however we cannot rely on constructions such as the HMAC (Krawczyk et al., 1997) due to memory constraints on the sensor, the only alternative being the CBC-MAC which can be easily derived from the symmetric cipher that we use.

Hardware constraints for in-wheel sensors were also taken into consideration. For this type of application, lightweight block ciphers are suited since they require low amounts of memory and processing power. The block cipher family of our choice is SPECK (Beaulieu et al., 2015), released by NSA and optimized for performance in software implementations.

3.3 Protocol description

Data packet format. In the available implementation examples, the Microchip sensor uses a data packet with the structure from Figure 7 (i). The ID field has three bytes, comparable to real sensors as shown in our experimental study. Pressure and temperature data are both two-byte fields and there is another one-byte field for sending useful flags such as battery status, giving a total of 8 bytes that are sent by the sensor.

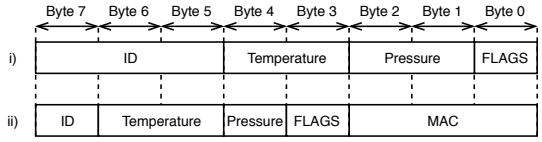

Figure 7: Structure of a data packet from TPMS

Figure 7 (ii) shows the data packet format used in our implementation once the security element is added. In order to achieve data integrity, a three-byte MAC field is added. To avoid increasing the power consumption during transmissions, the ID and pressure fields are reduced in size so that the final length of the data packet remains 8 bytes. Because the ID is encrypted and the message is authenticated (with a session key), using only one byte for the ID field is sufficient and even if collisions occur in the ID they are further mitigated by checking the larger MAC.

Freshness. In our approach, we achieve message freshness by using Counter mode for message encryption. This method prevents replay attacks and can be implemented with ease on low-end embedded systems. Considering that for the smallest SPECK variant i.e. SPECK32_64 the counter has the size of 32 bits and that TPMS sensors send data once a few minutes, capturing two messages encrypted with the same counter value is unlikely.

Confidentiality and integrity. The proposed protocol ensures confidentiality and integrity of TPMS RF transmissions by encrypting the messages and concatenating a three-byte MAC. Depending on the implementation requirements, we distinguish between two different scenarios. The first scenario corresponds to the situation in which the message length exceeds the block size (of the block cipher), thus an additional block is required for computing the MAC, i.e., CBC-MAC. This scenario is illustrated in Figure 8. In the second scenario, the entire message fits in the block cipher thus the MAC can be computed by simply encrypting the ciphertext with the authentication key. The second situation is illustrated in Figure 9.

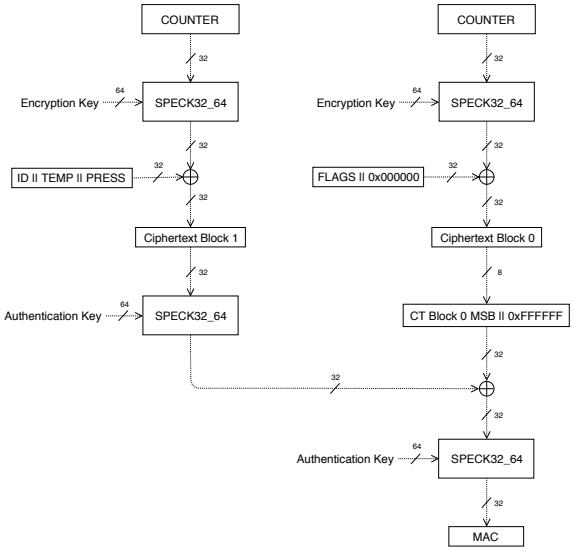

Figure 8: Encryption with SPECK32_64

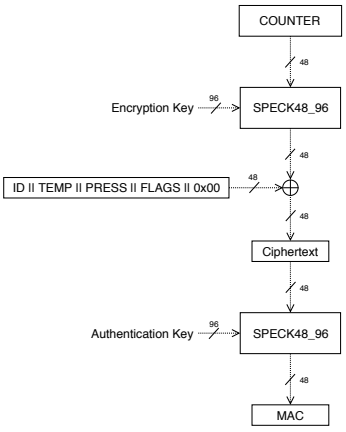

Figure 9: Encryption with SPECK48_96

Table 3: Execution time of Speck variants

Block size (bits)	**Key size** (bits)	**Execution time** (µs) S12XF512	PIC16F684
32	64	188.6	2693
48	72	198.6	3854
48	96	207.6	4044
64	96	273.6	5494
64	128	284.0	5697
96	96	301.8	7934
96	144	312.2	8217

Table 4: Memory utilization for PIC16F684

Block size (bits)	**Key size** (bits)	**Flash** (bytes)	**RAM** (bytes)	**EEPROM** (bytes)
32	64	44	7	44
48	72	60	9	66
48	96	60	9	69
64	96	76	11	104
64	128	76	11	108
96	96	112	15	168
96	144	112	15	174

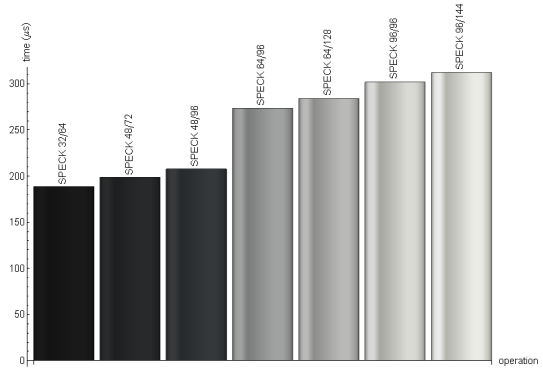

Figure 10: Encryption time with SPECK on S12

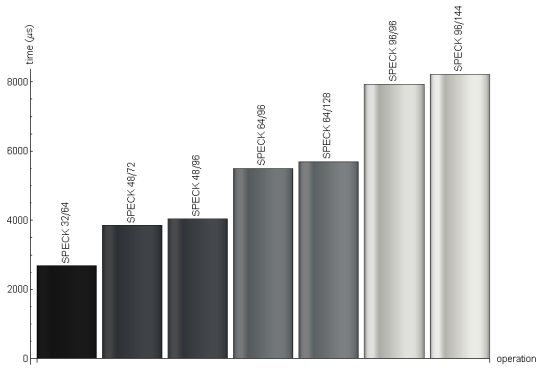

Figure 11: Encryption time with SPECK on PIC

3.4 Performance evaluation

Execution time. We now evaluate the performance of the employed platforms in executing the underlying cryptographic algorithms. Table 3 presents the execution times of different variants of SPECK (various combinations of block and key size) on the S12XF512 and PIC16F684. Figures 10 and 11 give a graphic summary of the encryption time on both of the controllers.

Memory utilization. Given the reduced memory available on sensor devices we address the issue of memory consumption with a memory-optimized implementation of SPECK on the PIC16F684. Memory optimization for SPECK is presented in (Beaulieu et al., 2015) as a trade-off between FLASH and RAM consumption. Our objective is to minimize both program and data memory usage by optimizing the algo-

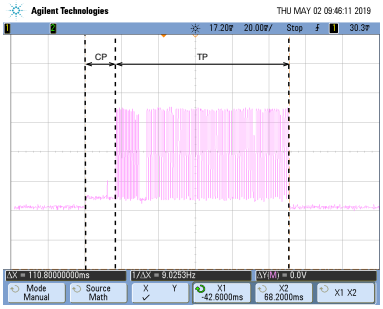

Figure 12: Power consumption: no security

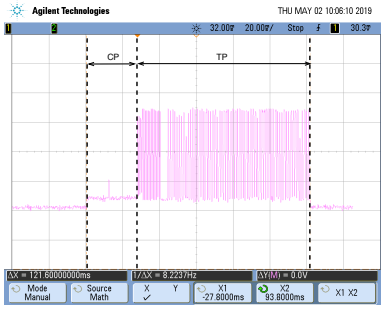

Figure 13: Power consumption: SPECK 32/64

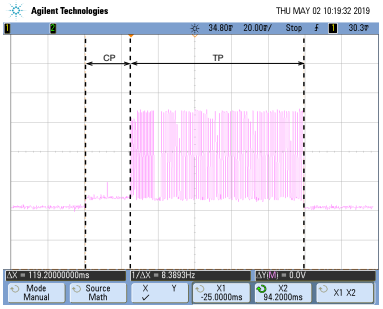

Figure 14: Power consumption: SPECK 48/96

rithms in assembly and taking advantage of the available EEPROM memory. The encryption algorithm is straightforward and there is little space for optimization. In contrast, the key expansion stage which is performed before every encryption can be skipped. During this stage, the secret key for encryption is expanded into round keys to be used in the encryption stage. By pre-expanding and storing the round keys in EEPROM, the key expansion stage is avoided, thus reducing the amount of RAM and FLASH memory used. Table 4 lists the memory consumption of our implementations of various SPECK variants on the PIC16F684. The last column is fully represented by the round keys stored in EEPROM.

Power consumption. Due to their in-wheel positioning, TPMS sensors are powered by batteries. Usually, the expected time before battery drain is up to seven years, thus minimizing energy increase is mandatory. Figure 12 illustrates the power consumption plot of Microchip's TPMS sensor, before adding security measures. After augmenting regular data transmission with the security protocol, based on both SPECK32 64 and SPECK48 96 block ciphers, the power consumption graph modifies as depicted in Figure 13 and Figure 14 (note that the computational time expands). The overall power consumption can be divided into two major phases, i.e., Computation Phase (CP) and Transmission Phase (TP).

The computational phase CP starts when the processor exits low-power mode and during this phase, sensor measurements, encryption and MAC computations are performed. Peak value is reached when the actual sensor is powered during pressure and temperature measurements and can be visually observed as a spike in all plots. The power consumption is dictated by the processor's operational power requirement and the average instantaneous value measured by us was 402 micro-watts. Comparing the CP durations we find increases of 65.8% in the SPECK32_64 based implementation and 50% in the variant based on SPECK48_96 (note however that power consumption is small compared to the transmission phase TP). Using a larger block, i.e., 48 vs. 32, to avoid an additional encryption block leads to lower computational time thus to lower power consumption. The second phase (TP) is marked by a dramatic increase of the plot amplitude. During this phase the message is sent via RF thus the antenna must be powered. By performing again the same measurement for the average power consumption, we obtain 1284 micro-watts. Because we used the same packet length of 8 bytes, the duration of TP is the same in all three cases. Using calculus to approximate the power consumption penalty for both cases, we estimate an overall increase of 3.3% for the SPECK32_64 variant and 2.5% for the SPECK48_96 variant. This shows that our security protocol does not impact the battery life significantly.

4 Conclusion

In this work we have analyzed the security of low-end vehicle sensors and proposed a cryptographic protocol that is able to assure both privacy and security. To prove feasibility, we made a proof-of-concept implementation on a TPMS kit from Microchip. Our protocol makes use of the lightweight SPECK cipher and assures security by encrypting the data-field and by using a CBC MAC. Performance results are discussed for several version of the cipher. Since TPMS systems usually rely on internal batteries that cannot be changed, we also present results on energy consumption which prove that the solution is not prohibitive w.r.t. small batteries on such sensors. Similar lightweight security mechanisms can be used for low-end RLS sensors. Our after-market RLS sensor was not programmable, but it has superior characteristics to the TPMS sensor that we used and thus the proposed solution can be deployed in a straight-forward manner. While current RLS sensors are wired to LIN bus, we argue that RF communication in RLS may become a future option to reduce wiring.

Acknowledgement. We thank the reviewers for helpful comments that improved our work. This work was supported by a grant of Ministry of Research and Inovation, CNCS-UEFISCDI, project number PN-III-P1-1.1-TE-2016-1317, within PNCDI III (2018-2020).

REFERENCES

AUTOSAR (2015a). *Specification of Crypto Abstraction Library*, 4.2.2 edition.

AUTOSAR (2015b). *Specification of Crypto Service Manager*, 4.2.2 edition.

AUTOSAR (2017). *Specification of Secure Onboard Communication*, 4.3.1 edition.

Beaulieu, R., Treatman-Clark, S., Shors, D., Weeks, B., Smith, J., and Wingers, L. (2015). The simon and speck lightweight block ciphers. In *2015 52nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, pages 1–6. IEEE.

Checkoway, S., McCoy, D., Kantor, B., Anderson, D., Shacham, H., Savage, S., Koscher, K., Czeskis, A., Roesner, F., Kohno, T., et al. (2011). Comprehensive experimental analyses of automotive attack surfaces. In *USENIX Security Symposium*. San Francisco.

Ergen, S. C. and Sangiovanni-Vincentelli, A. (2017). Intravehicular energy-harvesting wireless networks: Reducing costs and emissions. *IEEE Vehicular Technology Magazine*, 12(4):77–85.

Ishtiaq Roufa, R. M., Mustafaa, H., Travis Taylora, S. O., Xua, W., Gruteserb, M., Trappeb, W., and Seskarb, I. (2010). Security and privacy vulnerabilities of in-car wireless networks: A tire pressure monitoring system case study. In *19th USENIX Security Symposium, Washington DC*, pages 11–13.

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

Krawczyk, H., Bellare, M., and Canetti, R. (1997). Hmac: Keyed-hashing for message authentication. Technical report.

Miller, C. and Valasek, C. (2014). A survey of remote automotive attack surfaces. *Black Hat USA*.

Solomon, C. and Groza, B. (2015). Limon - lightweight authentication for tire pressure monitoring sensors. In *1st Workshop on the Security of Cyber-Physical Systems (affiliated to ESORICS 2015)*.

Takahashi, J., Aragane, Y., Miyazawa, T., Fuji, H., Yamashita, H., Hayakawa, K., Ukai, S., and Hayakawa, H. (2017). Automotive attacks and countermeasures on lin-bus. *Journal of Information Processing*, 25:220–228.

Toth, A. (2014). Method and system for monitoring a parameter of a tire of a vehicle. EP Patent App. EP20,120,464,019.

van Zyl, P., Goethem, S. v., Jansen, S., Kanarchos, S., Rexeis, M., Hausberger, S., and Smokers, R. (2013). *Study on tyre pressure monitoring systems (tpms) as a means to reduce light-commercial and heavy-duty vehicles fuel consumption and CO2 emissions*. Delft: TNO.

Xu, M., Xu, W., Walker, J., and Moore, B. (2013). Lightweight secure communication protocols for in-vehicle sensor networks. In *Proceedings of the 2013 ACM workshop on Security, privacy & dependability for cyber vehicles*, pages 19–30. ACM.

Yan, C., Xu, W., and Liu, J. (2016). Can you trust autonomous vehicles: Contactless attacks against sensors of self-driving vehicle. *DEF CON*, 24.