INTL JOURNAL OF ELECTRONICS AND TELECOMMUNICATIONS, 2023, VOL. 69, NO. 1, PP. 83–88
Manuscript received May 27, 2022; revised February, 2023. DOI: 10.24425/ijet.2023.144335

Hardware Implementation of an Enhanced Security- and Authentication-Related Automotive CAN Bus Prototype

Asmae Zniti, and Nabih EL Ouazzani

***Abstract*—In this paper a new security technique aiming to ensure safe and reliable communications between different nodes on an automotive Controller Area Network (CAN) is presented. The proposed method relies on a robust authentication code using Blake-3 as a hash algorithm within an adapted structure that includes a monitor node. A prototype is implemented and run effectively to perform hardware simulations of real case-based security problems of automotive embedded CAN systems. As a result, data transfer can take place on a newly enhanced CAN bus according to the standard protocol without being intercepted nor tampered with by unauthorized parties thereby highlighting the effectiveness of the proposed technique.**

***Keywords*—CAN; automotive security; attack; authentication**

I. Introduction

NOWADAYS automotive CAN bus-based networks are implemented with the objective of controlling all vehicle functions. Incorporated devices such as, controllers, sensors and actuators, which are supervised by such a network generally fall into two categories: safety and comfort [1], [2].

Although a CAN bus is actually regarded as a simple and effective monitoring system, it basically lacks fundamental security features and can therefore be subject to malevolent software attacks [3], [4]. As a matter of fact, over the past few years several attacks have been reported thereby pointing out the extent of security and vulnerability problems which designers have yet to handle.

In 2015, security researchers demonstrated how to take remote control of a Tesla Model S simply by plugging a device into the OBD-II port [5]. The attack allowed the researchers to remotely unlock doors, open the trunk, and even drive away with the vehicle. In 2016, another group showed how a Tesla Model S could be wirelessly hacked into and various systems such as brakes, steering, and door locks could be commandeered [6]. The attack only required about 90 seconds of physical access to the car in order to upload a malicious code onto the car CAN bus. While the previous attacks required physical access to the vehicle, there have been other instances reported of entirely remote attacks against vehicles by exploiting vulnerabilities in the CAN bus [7], [8], [9]. In 2010, a group of researchers showed how they could remotely disable the brakes of a Jeep Cherokee simply by sending malicious messages over the CAN bus [10].

With the development of new wireless interfaces such as vehicle-to-vehicle and vehicle-to-infrastructure, the number of opportunities for wireless attacks will undoubtedly increase, thus highlighting how extremely dangerous future threats can be [11], [12], [13].

This paper aims to investigate the security problems of the CAN protocol and point out the vulnerabilities that modern cars can be susceptible to. After identifying the CAN limitations, a solution based on authentication concepts [14] will be described. Several hash algorithms are now known and available thereby offering multiple options for securing and protecting data especially in embedded systems. The possibility of incorporating such algorithms into CAN bus networks opens up new and robust avenues for automotive industry designers. The rest of the paper is structured as follows: In Section II, a general overview of the CAN protocol is given. Section III demonstrates CAN bus security shortcomings through a typical application. In Section IV, the entire process is broken down into different stages whose conditions are detailed along with a comprehensive application. The work is summarized and conclusions are drawn in Section V.

II. A Brief CAN Bus Introduction

A. Bus Data Frame

The CAN protocol [15] has two versions that differ based on the message identifier length: CAN 2.0A (the standard CAN) with an 11-bit identifier and CAN 2.0B (extended CAN) with a 29-bit identifier [16]. A CAN bus data frame whose structure is shown in Fig. 1. is a data packet that contains information about the sender, the receiver, and the message itself. The frame consists of several fields, each of which has a specific purpose. The structure of such a frame is as follows:

- SOF (Start of Frame): This is a single bit that indicates the beginning of the frame. It is always set to 0.
- The header contains:
  Frame identifier: an 11-bit or 29-bit value that uniquely identifies the message within the network. All devices on the network use this ID to determine whether they should receive or ignore the message.
  RTR bit: Remote Transmission Request which defines a data frame if it is set to 0 and a request frame of data if it is set to 1.

Authors are with Faculty of Sciences and Technology (FST), University Sidi Mohamed Ben Abdellah, Signals Systems and Components Laboratory (LSSC), Fez, Morocco (e-mail: znitiasmae@gmail.com, nabih.elouazzani@usmba.ac.ma).

DLC field: Data Length Code which indicates the size in bytes of the transmitted data.

- Data field: This is where the actual message data is stored. The size of this field depends on the DLC.
- CRC checksum: This is used to detect errors in transmission. If the receiving device calculates a different CRC from the one that is sent, then it knows that an error has occurred and can take appropriate actions.
- The ACK (Acknowledgment) field, which acknowledges receipt of the frame and allows the sender to know if its message is received correctly by at least another ECU (but does not guarantee that it is necessarily the intended recipient initially)
- The EOF (End of Frame) field followed by an intermission frame, which designates the minimum number of bits to pass before the transmission of another message can start.

SOF	Header	Application data	CRC field	Ack	EOF	Intermission
1 bit	**18 bits Standard CAN (2.0A)** **36 bits Extended CAN (2.0B)**	**0..8 bytes**	**15 bits**	**3 bits**	**7 bits**	**3 bits**

Fig. 1. Data frame structure

B. Communication Protocol

CAN bus nodes mainly consist of electronic control units (ECU) which are equipped with software codes whose aim is to deal with all aspects of data transmission under a specific communication protocol. An ECU is primarily a microprocessor which contains a CAN controller used to support data link layer functions and a CAN transceiver used for physical layer functions such as frame delivery, error detection and correction, as well as other data link layer tasks.

Each individual node on a CAN bus has the ability to read and write data freely throughout the network. There is no ‘Master-Slave’ concept, that is, any node on the network can initiate a communication at any time. Each node has an arbitration ID that is used to identify messages and indicate priority when multiple nodes try to send frames at the same time [17].

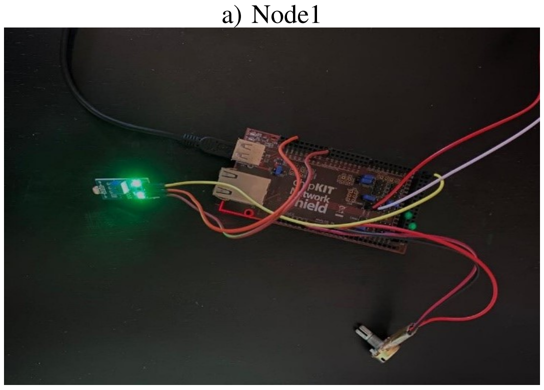
Depending on the arbitration ID, a CAN node decides whether to accept a frame or ignore it. When a CAN node correctly receives a message, it tags an ACK bit onto the end of the transmission. The node that sends this message receives this ACK bit and then removes it from the bus. When a CAN node detects an error in a received message, it sends out an ERROR frame instead of an ACK bit. The ERROR frame provides information in relation to the type of error. There are several error detection mechanisms built into the CAN system using an error frame made up of 6 consecutive dominant bits (error flags). When an ECU detects an error, it emits an error frame to warn the other members of the network. The sender of the erroneous frame must then resend it in accordance with the next arbitration step.

III. Examples of the CAN Bus Vulnerability

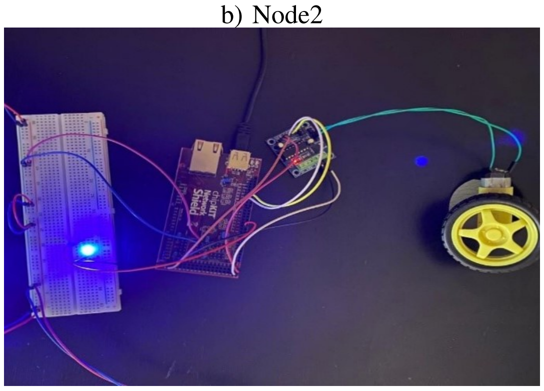
A. Hardware Platform

A hardware prototype is designed with respect to the CAN bus protocol and built on a ChipKIT Max32 board tied through point-to-point connections to various devices, sending and receiving signals similar to those used in automotive systems. For distributing the generated data, these cards will be interconnected by means of a CAN Network Shield. The Max32 board is a 32-bit Microchip® PIC microcontroller that takes advantage of the PIC32MX795F512L and runs at up to 80 MHz with 512KB of flash program memory and 128KB of SRAM data memory. In addition, the processor provides dual CAN controllers. These CAN controllers in combination with two Microchip MCP2551 CAN transceivers on the Network Shield allow the Max32/Network Shield to operate on one or two independent CAN networks. The Max32 can be programmed by using Mult-Platform IDE (MPIDE).

a) Node1



b) Node2



c) Node3

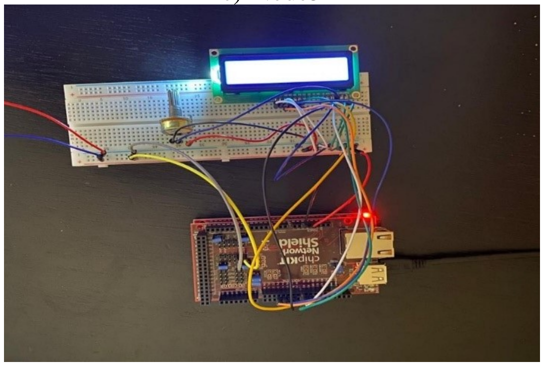


Fig. 2. Hardware set-up

An automotive communication system is simulated by means of transmitting various signals between three nodes described as follows:

Node1: This controller has a 10k- potentiometer connected to its analogue pin 0, which in turn causes the voltage on this pin to vary between 0 and 5V. The generation of this signal aims to control the speed of a DC motor connected to node2, simulating the operation of an accelerator pedal. This speed control will be achieved by creating and sending a PWM (Pulse-Width Modulation) signal as a frame on the CAN bus, with the id 0x10. This controller also has an LDR-type light sensor connected to its analogue input 1. LDRs are light-dependent resistors, meaning that their resistance decreases when the amount of light they are exposed to increases. By monitoring the output of an LDR sensor, a car's ECU can automatically adjust the brightness of its headlights, tail lights, and interior lights based on the current conditions. This helps improve visibility for drivers and makes it easier to see potential hazards on the road. In the proposed platform the LDR signal is sent on the CAN bus labelled 0x20 and is used to control the light brightness of the LED on node2. Fig. 2-a shows node1 elements.

Node2: This controller receives signals generated by node1 to control the motor speed and LED lights. Fig. 2-b is a photo of node2.

Node3: This ECU uses the signals identified by 0x10 and 0x20 to display on the LCD the potentiometer value and the light signal transmitted by node1. A photo of node3 is presented in Fig. 2-c.

B. *Attack Scenarios*

Two different attacks on the network of Fig. 2 are carried out, and described as follows:

- In the first scenario, the DC motor is targeted by adding malicious frames with 0x10 as ID by falsifying the potentiometer value to 0, and as a result, the DC motor speed decreases to 0. This issue shows that in the case of a vehicle-under-attack scenario, when a motor node reads the value 0, the car speed drops to 0 km/h leading to its steering and braking being blocked, and the engine perhaps stalling until the end of the attack. The fact that the motor breaks down suddenly will cause the wheels to lock up and the car to likely skid out of control.
- With regard to the second attack, an LDR sensor, which is usually inserted to detect daylight and then make the car's headlights turn on or off accordingly, is subject to a software alteration. As a result, the headlights switch off while it is dark causing the driver to lose visibility and putting passengers lives in jeopardy as well.

Table I gives a recap of the targeted elements and the resulting risks detailed previously.

This vulnerability comes from the fact that the CAN bus uses a broadcast medium, allowing every node to see all messages. So, an attacker could read sensitive information, send false messages or tamper with data to manipulate the vehicle functions and disturb the driver. Therefore, it is highly required to add an enhanced level of security onto the current protocol in order to avoid such eventualities

TABLE I
DESCRIPTION OF THE ATTACK SCENARIOS

Scenario	Attack	Vulnerability	Action	Target	Result
Motor DC	Uploading malicious code	CAN bus protocol without message authentication	Read /spoof	Electronic control units (ECU)	Motor disabled
Lights system					Lights system malfunctioning

IV. PROPOSED AUTHENTICATION TECHNIQUE

A. *Software Coding Operations*

In order to overcome the vulnerability issue of CAN bus systems, a new two-phase method requiring an additional node is presented and detailed within this section. This node is mainly dedicated to performing cryptographic operations and it is called the monitoring node [18]. The two phases are as follows:

- First phase: Node authentication

The monitoring node generates and sends a random data called nonce, which will be used as an address of the program section meant to be authenticated. This nonce must be processed and the results sent back to the monitor. If any node does not respond or sends incorrect data, it must be concluded that a malicious node is on the network. Fig. 3 shows all steps of this operation. Individual participating node codes are pre-shared with the monitor and introduced into the following formula (1) to generate the hash result of several digests 'KEY i'.

$$KEYi = hashfunction(MSG||NONCE) \qquad (1)$$

where:
KEY i: the authentication key of the sender node i.
MSG: a part of the program code of the sender node i.
NONCE: a random seed.

KEY i will be used as a cryptographic key in the second phase. This process aims to set a new key for each communication session and to dismiss the previously used ones. As a result, replayed attacks can be avoided in case a hacker manages to record traffic from earlier attempts.

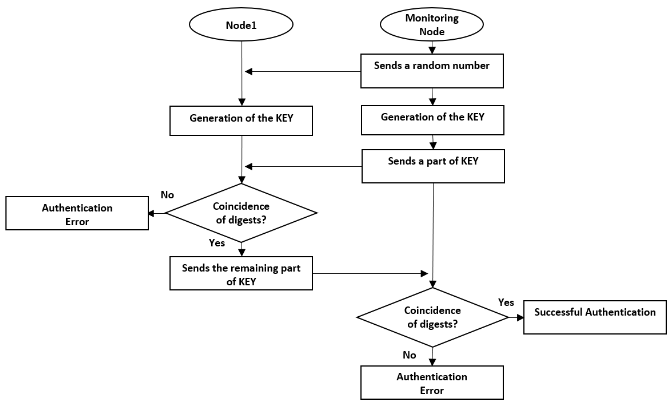


Fig. 3. Phase1 of the proposed security system protocol

If the authentication fails, the monitoring node sends a warning signal to the driver indicating that a compromised node is on the network. A possible solution which can easily be integrated into the process, consists in downloading the legitimate code from the monitor onto the previously compromised and newly cleared node.

- Second phase: Frame authentication

This phase has the purpose of identifying the legibility of data frames circulating on the bus, thus allowing the sender node to generate two frames one after another as indicated in Fig. 4. First, the application data frame is sent on the CAN bus followed up by the authentication data frame. Fig. 5 illustrates the procedure.

The following equation (2) is used to generate the authentication data:

$$MACi = hashfunction(IDi, Di, FCi, KEYi) \quad (2)$$

where:
IDi: CAN-ID.
Di: message i data.
FCi: Complete monotones counter for message i.
KEY i: the encryption key for the sender node i.

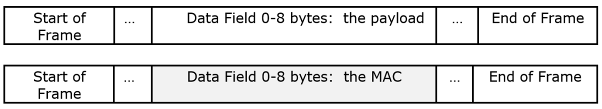


Fig. 4. CAN frames involved in the proposed technique

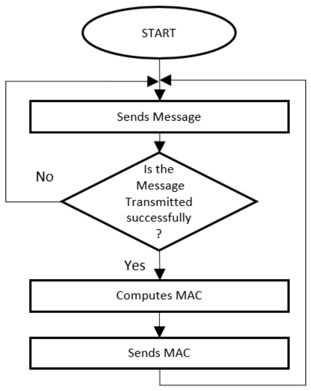


Fig. 5. Sender node

First, the monitoring node Fig. 6 receives data from the sender, and immediately generates a calculated cryptographic MAC, called MacCalc. It then receives the authentication frame and compares the received and the calculated MACs. If at least one digest is not valid, the message is discarded and regarded as an attack.

According to the new method, a receiver Fig. 7 is set to sort processed data right after the monitor has sent the confirmation of a message's legitimacy. Obviously, if an attack occurs, the monitoring node alerts the receivers to dismiss the malicious message through a warning frame transmitted with the highest priority. However, the message is consequently considered as legitimate if no warning is sent within a certain period of time (about 70 μs).

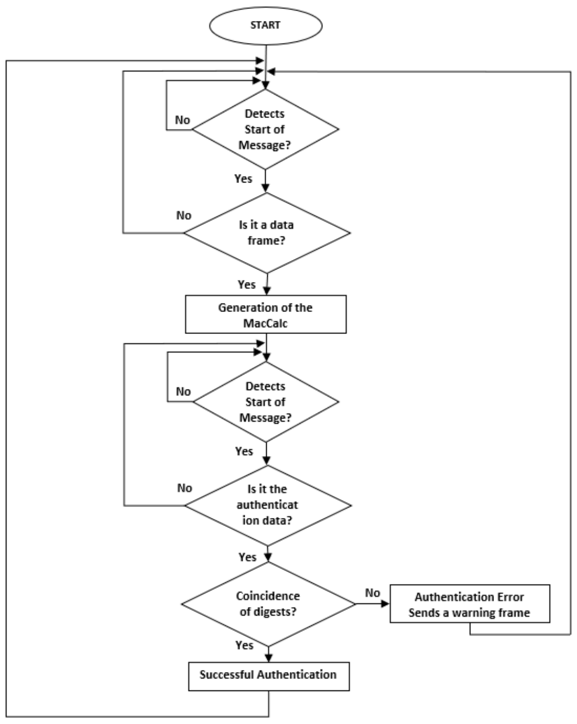

START
No
Detects Start of Message?
Yes
No
Is it a data frame?
Yes
Generation of the MacCalc
No
Detects Start of Message?
Yes
No
Is it the authenticat ion data?
Yes
Coincidence of digests?
No
Authentication Error Sends a warning frame
Yes
Successful Authentication

Fig. 6. Monitoring node

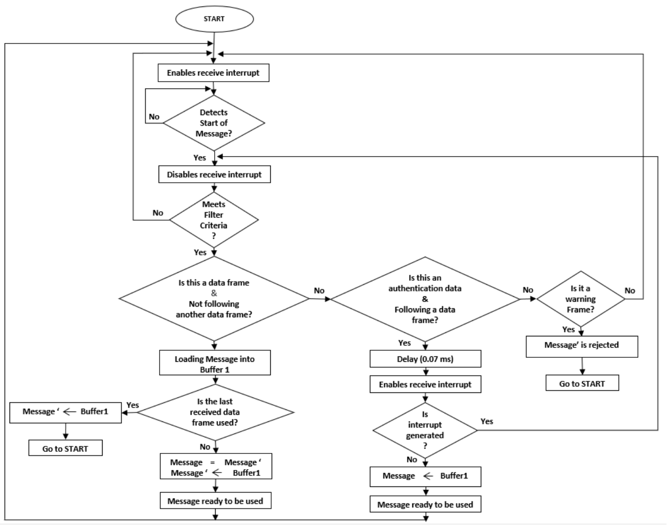


Fig. 7. Receiver ode

B. BLAKE-3 Hash Algorithm

Generally, a hash algorithm is applied to calculate two types of digests with respect to the following entries:

- A code which is implemented on a node participating in the CAN communication.
- A message to be sent.

Blake-3 [19] hashing algorithm was designed and optimized for parallel computation and speed on modern CPU architectures. It has a very compact design with only three rounds of compression functions while maintaining the security level equivalent to BLAKE2b's [20] which is one of the most widely used cryptographic hash algorithms today.

The basic idea behind Blake-3 can be summarized as follows: Blake-3 takes an input of arbitrary length, and outputs a digest of 256 bits. The algorithm divides the input into equally sized blocks. For each block, it applies three rounds of compression functions to generate the final hash output. This is different from other cryptographic hashing algorithms where more rounds are needed such as SHA-256 [21] (64 rounds) or BLAKE2b (12 rounds). Blake-3 is also designed to be parallelized. For instance, the three rounds can be run on a multiple core CPU architecture. The final hash output of each block is used as an input for the next one, if any. Unlike traditional cryptographic hashing functions that compute digest by simply concatenating all blocks' outputs, Blake-3 computes it with an extra permutation step which makes its design more secure against side channel and birthday attacks. As far as this technique is concerned, The Blake-3 has been chosen as the cryptographic hashing algorithm for mainly three reasons:

- Blake-3 is a very fast algorithm and can be used in parallel, which makes it even faster.
- The hash function is collision-resistant in that two different inputs never produce the same output.
- Blake-3 ensures a high level of security. There have been no known attacks reported against it so far.

C. Hardware Implementation

Fig. 8 shows a set-up of a four-node hardware prototype. The implementation includes three Max32 board-based participating nodes along with a Zynq-7000 Artix-7 FPGA-based monitor which offers greater computational speed and programming option advantages over a 32-bit microcontroller. The entire system configuration and implementation have been achieved by means of the Vivado tool platform.

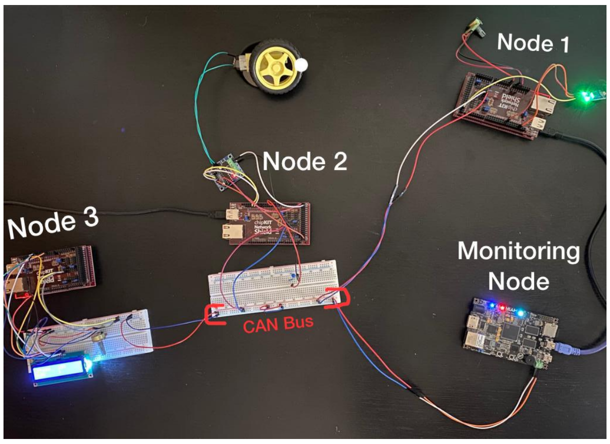


Fig. 8. Experimental prototype

As an application, time-related performances during a data frame processing phase are investigated and then compared in this real case-type hardware simulation. Indeed, the processing times required in both secured and corrupted communications are measured as well as the time interval needed by an under-attack node to appropriately handle a malevolent message. Results are detailed in Table II and III. Various malicious messages are sent over the bus periodically as a way of setting up attacks on nodes 2 and 3. The first message is referenced to as a falsified potentiometer value while the second one provides the state of LDR sensor with a false content.

TABLE II
DESCRIPTION OF THE ATTACK SCENARIOS

Instant	Sender			Monitor node	Receiver	Required time (ms)
	Node	ID	Data Field			
t1	1	0x10	30	-Data = 30 -Generation of the MacCalc	-Data = 30	5.274
t2	1	0x01	MACi	-Receives MACi -Compares MACi and MacCalc -Successful authentication	-Receives MACi -Comparison delay of 0.07 ms -Enables receive interrupt -Uses data(30)	
t3	1	0x20	light	-Data = light -Generation of MacCalc	-data= light	5.26
t4	1	0x02	MACi	-Receives MACi -Compares MACi and MacCalc -Successful authentication	-Receives MACi -Comparison delay of 0.07 ms -Enables receive interrupt -interrupt(t5)	
t5	1	0x10	40	-Data = 40 -Generation of MacCalc	-Uses data (light) -Data = 40	
...	...	...	...	...	...	...

TABLE III
DATA PROCESSING DURING ATTACKS

Instant	Sender			Monitor node	Receiver	Required time (ms)	
	Node	ID	Data Field				
t1	1	0x10	50	-Data = 50 -Generation of MacCalc	Data = 50	5.262	
t2	1	0x01	MACi	-Receives MACi -Compares MACi and MacCalc) -Successful authentication	-Receives MACi -Comparison delay of 0.07ms -Enables receive interrupt -Interrupt (t3)		
t3	Attacker	0x10	00	-Data = 00 -Generation of the MacCalc	-Data = 00 -Uses data (50)	5.13	
t4	Attacker	0x01	MACi	-Receives MACi -Compares MACi and MacCalc) -Successful authentication	-Receives MACi -Comparison delay of 0.07ms -Enables receive interrupt -Interrupt (t5)		0.078
t5	Monitor node	0x00	0	(Sender)	-Receives warning frame -Rejects data (00)		
t6	1	0x20	light	-Data = light -Generation of MacCalc	-Data = light	5.257	
t7	1	0x02	MACi	-Receives MACi -Compares MAC and MacCalc -Successful authentication	-Receives MACi -Comparison delay of 0.07ms -Enables receive interrupt -Interrupt (t8)		
t8	1	0x10	60	-Data = 60 -Generation of MacCalc	-Data = 60 -Uses Data (light)		
t9	Attacker	0x20	Dark	-Rejects data and waits for the Authentication data	-Rejects data and waits for the Authentication data	0.024 between reception of frame and detection of error	

Tables II and III clearly illustrate that authentication operations require a computing time that can be considered as satisfactory with respect to automotive standards. According to the results, handling a message on the CAN bus takes approximately 5.27 ms and therefore meets the regulatory time constraints in a real-time environment. The first phase takes a delay of about 64.25 ms for each node. In the case of 70 ECUs, which is known as the maximum number of units within a vehicle, it lasts almost 4.4 s and it is consequently suitable for in-vehicle applications. The major advantage of this approach lies in the fact that it needs neither hardware modifications nor changes in the CAN bus protocol. A simple node, consisting of a monitoring controller, can be easily fit into this existing architecture for cryptographic purposes without reducing payload sizes in data frames.

V. Conclusion

Several aspects of the automotive CAN bus vulnerability have been demonstrated in this paper, through some real-case hardware simulations. As a result, ECU functions and related operations have been disturbed and canceled pointing out the lack of security and data protection within such systems.

A two-phase security technique which aims to ensure authentication and guarantee protection of automotive embedded networks has been proposed. The core structure relies on a central node that monitors data traffic on in-vehicle CAN bus systems.

The first phase of the procedure is structured to verify the authenticity of an ECU program code along with sharing a generated key with all the other participating nodes. The second phase deals with identifying the frame legibility through assigning a MAC code to each message circulating on the bus.

A Blake3-based hash algorithm has been chosen, due to its reduced calculation time, and integrated into the software platform to achieve the necessary cryptographic operations in terms of node authentication and key generation. Several hardware attack simulations have been carried out and dealt with adequately within a time frame of 5.27ms which is readily acceptable in such systems. The prototype has shown excellent resistance to software attacks and blocked all malicious attempts while preserving the initial functions of all ECU-based nodes.

Finally, it should be pointed out that the proposed hardware platform and the software code can be easily implemented, and run on the existing structure; therefore no new architecture is needed.

References

[1] P. Mundhenk, "Security for Automotive Electrical / Electronic (E / E) Architectures", Cuvillier Verlag, Göttingen, Germany, 2017.

[2] "'ECU' is a Three Letter Answer for all the Innovative Features in Your Car: Know How the Story Unfolded", Embitel, 2017. [Online]. ://www.embitel.com/blog/embeddedblog/automotive-control-units-development-innovationsmechanical-to-electronics [Accessed: 23-May-2018].

[3] R. Buttigieg, M. Farrugia, and C. Meli, "Security Issues in Controller Area Networks in Automobiles", in 18th international conference on Sciences and Techniques of Automatic Control Computer Engineering, 2017, pp. 21–23. https://doi.org/10.1109/STA.2017.8314877

[4] P. Carsten, T. R. Yampolskiy, and J.T. Macdonald, "In-vehicle networks: Attacks, vulnerabilities, and proposed solutions", In Proceedings of the 10th Annual Cyber and Information Security Research Conference. Oak Ridge. (p. 1). (2015). https://doi.org/10.1145/2746266.2746267

[5] M. Rogers, "How we hacked a Tesla Model S in 30 minutes". Black Hat, 2015. https://www.blackhat.com/us-15/briefings.htmlhow-we-hacked-a-tesla-model-s-in-30-minutes

[6] S. Nie, L. Liu, and Y. Du, "Free-fall: hacking tesla from wireless to can bus", Briefing, Black Hat USA, pp. 1–16, 2017. https://doi.org/10.1145/2746266.2746267

[7] A. Zniti, and N. E. Ouazzani, "Implementation of a bluetooth attack on controller area network (CAN)", *Indonesian Journal of Electrical Engineering and Computer Science*. 21. 321. 10.11591/ijeecs.v21.i1.pp321-327, 2020. https://doi.org/10.11591/ijeecs.v21.i1.pp321-327

[8] S. Woo, H. J. Jo, and D. H. Lee, "A Practical Wireless Attack on the Connected Car and Security Protocol for In-Vehicle CAN", *IEEE Trans. Intell. Transp. Syst*, vol. 16, no. 2, pp. 993–1006, 2015. https://doi.org/10.1109/TITS.2014.2351612

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

[10] S. Checkoway, D. McCoy, B. Kantor, D. Anderson, H. Shacham, S. Savage, K. Koscher, A. Czeskis, F. Roesner, and T. Kohno, "Comprehensive Experimental Analyses of Automotive Attack Surfaces", Proc. USENIX Security Symposium, 2011.

[11] J. E. Siegel, D. C. Erb, and S. E. Sarma, "A survey of the connected vehicle landscape-architectures, enabling technologies, applications, and development areas," IEEE Transactions on Intelligent Transportation Systems, vol. 19, no. 8, pp. 2391–2406, 2018. https://doi.org/10.1109/TITS.2017.2749459

[12] J. Liu, S. Zhang, W. Sun, and Y. Shi, "In-vehicle network attacks and countermeasures: challenges and future directions", *IEEE Network*, vol. 31, no. 5, pp. 50–58, 2017. https://doi.org/10.1109/MNET.2017.1600257

[13] Z. El-Rewini, K. Sadatsharan, D. F. Selvaraj, S. J. Plathottam, and P. Ranganathan, "Cybersecurity challenges in vehicular communications", Veh. Commun. 2020. https://doi.org/10.1016/j.vehcom.2019.100214

[14] K.S. Mohamed, "Cryptography Concepts: Integrity, Authentication, Availability, Access Control, and Non-repudiation", In New Frontiers in Cryptography: Quantum, Blockchain, Lightweight, Chaotic and DNA, Springer International Publishing: Cham, Switzerland, 2020, pp. 41–63. https://doi.org/10.1007/978-3-030-58996-7$_3$

[15] R. Bosch GmbH, "CAN Specification, version 2.0", 1991.

[16] Y. Lv, W. Tian and S. Yin, "Design and Confirmation of a CAN bus Controller Model with Simple User Interface", Fifth International Conference on Instrumentation and Measurement, Computer, Communication and Control (IMCCC), Qinhuangdao, 2015, pp. 640-644, https://doi.org/10.1109/IMCCC.2015.140

[17] M. Di Natale, H. Zeng, P. Giusto, and A. Ghosal, "Understanding and using the controller area network communication protocol: theory and practice", Springer Science Business Media, NY, 2012. https://doi.org/10.1007/978-1-4614-0314-2

[18] A. Zniti and N. El Ouazzani, "Improvement of the Authentication on In-Vehicle Controller Area Networks", Embedded Systems and Artificial Intelligence, vol. 1076, pp. 23-32, 2020. https://doi.org/10.1007/978-981-15-0947-6$_3$

[19] S. Neves, J. O'Connor, J.P. Aumasson, and Z. Wilcox-O'Hearn, "BLAKE3: One function, fast everywhere", GitHub, 2020. https://blake3.io

[20] J. P. Aumasson, S. Neves, Z. W. O'Hearn, and C. Winnerlein, "BLAKE2: Simpler, smaller, fast as MD5", In Applied Cryptography and Network Security, 2013, pp.119–135. https://doi.org/10.1007/978-3-642-38980-1$_8$

[21] D.Rachmawati, J. Tarigan, and A. Ginting, "A comparative study of message digest 5 (md5) and sha256 algorithm Journal of Physics", Conference Series, Vol. 978, IOP Publishing (2018). https://doi.org/10.1088/1742-6596/978/1/012116