Efficient Physical Layer Key Agreement for FlexRay Networks

Pal-Stefan Murvay and Bogdan Groza

***Abstract*—The FlexRay protocol provides deterministic and fault-tolerant communication for automotive applications with stringent requirements regarding reliability and real-time performance. But the lack of intrinsic security mechanisms makes FlexRay vulnerable to spoofing and DoS attacks while existing features are not sufficient to assure reliable communication in the presence of adversaries. Since adding cryptography mandates for secret shared keys, providing secure key establishment techniques is critical in assuring the intended security objectives. In this paper we propose a backward compatible key agreement scheme for FlexRay communication based on intrinsic characteristics of the FlexRay physical layer. In particular, we determine that the symbol window is suitable for implementing a covert channel based on physical properties of the bus which allows us to bootstrap a secure session key between two nodes or to re-enforce existing shared keys. We evaluate and discuss the performance of software-based implementations of the proposed mechanism on two different automotive embedded platforms relying only on the application level.**

***Index Terms*—FlexRay, physical layer, key agreement.**

I. Introduction

Research on the security of in-vehicle environments was mostly focused on CAN (Controller Area Network) based communication. This does not come as a surprise since CAN is still the most widespread protocol used for in-vehicle communication. Nevertheless, the in-vehicle ecosystem also encompasses other communication protocols that should not be overlooked when considering automotive systems security. FlexRay is an automotive communication protocol employed by safety critical systems such as brake-by-wire, drive-by-wire or more advanced driver assistance systems.

Possible attacks on FlexRay communication were mentioned as early as 2004 along with the first concerns on in-vehicle security in general [1]. A first simulation of attacks on FlexRay was done by Nilsson *et al.* [2] while a more recent work provides a detailed view of the security of the FlexRay protocol and demonstrates a series of DoS (denial-of-service) and spoofing attacks on an actual test network [3]. An attacker that infiltrates the in-vehicle network and compromises a node of the FlexRay cluster can easily mount DoS and spoofing attacks on FlexRay communication. Compromising in-vehicle ECUs (electronic control units) has been proved to be feasible on modern automobiles by using the diagnostics port [4] and other on-board attack surfaces [5] or even by remote connection [6].

Securing FlexRay communication is important since manipulating it could leave safety critical ECUs unable of fulfilling their function or even be used to induce unwanted behavior that can jeopardize human safety. This risk can be circumvented by enhancing FlexRay communication with proper security mechanisms. However, a secure key establishing mechanism is required for achieving basic security objectives in a network. To the best of our knowledge the few works that cover the security of FlexRay networks do not provide a solution for key agreement or distribution that is specifically tailored for FlexRay clusters.

Contribution and addressed scenarios: In this paper, we propose a method for implementing key agreements in a FlexRay cluster that uses intrinsic characteristics of the physical layer. The principle that we use in this work is based on a key sharing mechanism that was previously proposed for CAN [7]. We adapt this mechanism and map the key agreement functionality to existing FlexRay specifications. To validate our proposal we implement the mechanism on two automotive grade embedded platforms by only using the application layer and FlexRay transceiver to support immediate software deployment. We also show that our approach is backwards compatible with existing FlexRay implementations and does not disrupt normal communication.

Fig. 1 is suggestive for the addressed scenarios. Several nodes are depicted on the bus sharing two bit-sequences (from which a secure secret-key can be derived) via a covert mechanism in the symbol window. There are two obvious scenarios that can benefit from the proposed mechanism. The first scenario is regular key exchange between nodes. Current standards for vehicle on-board communication do require secret shared keys

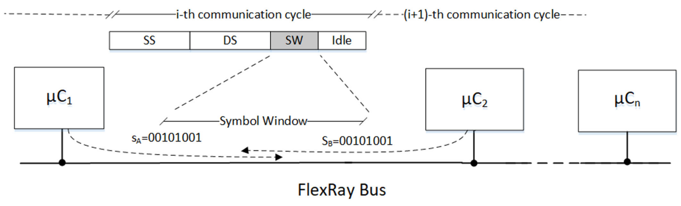

Fig. 1. Graphical representation of the addressed scenario with two nodes performing a key exchange during the symbol window.

Manuscript received February 25, 2020; accepted June 9, 2020. Date of publication June 12, 2020; date of current version October 13, 2020. This work was supported by the Romanian Ministry of Research and Innovation, CNCS - UEFISCDI, Project number PN-III-P1-1.1-PD-2016-1198, within PNCDI III. The review of this article was coordinated by Dr. X. Dong. *(Corresponding author: Pal-Stefan Murvay.)*

The authors are with the Department of Automation and Applied Informatics, Politehnica University of Timisoara, Timisoara 300223, Romania (e-mail: pal-stefan.murvay@aut.upt.ro; bogdan.groza@aut.upt.ro).

Digital Object Identifier 10.1109/TVT.2020.3002616

to communicate between peers, e.g., Section 5.2 in [8]. But the secure exchange of such secret keys requires the existence of a public-key mechanism, e.g., RSA or Diffie-Hellman. While function templates for such mechanisms are already specified in the AUTOSAR standard [9], public-key cryptography is computational expensive and moreover such a handshake requires for additional specific slots to be allocated in the static or dynamic segment of FlexRay. In contrast, the mechanism that we propose requires little computations, it is fully transparent and only requires minimum use of the dynamic segment for protocol bootstrapping. The second scenario of interest to us, calls for re-enforcing existing shared keys. Public key mechanisms can be used to exchange session keys, but rather than refreshing these keys with new handshakes (which will occupy additional space within the communication cycle), the covert channel that we create can achieve this in a transparent manner. Cryptographic ratchets can be used to increase the entropy of existing keys or to derive new keys between nodes.

Paper outline: The remainder of this paper is organized as follows. Section II presents related work on FlexRay security and physical layer key agreement mechanisms. Section III gives a background on the FlexRay protocol while Section IV describes the employed key sharing mechanism and its mapping to FlexRay. Next in Section V we present the proposed key agreement protocol before providing implementation details and evaluation results in section VI and VII. Finally, in Section VIII, we conclude our work.

II. Related Work

We discuss related work along two distinct lines: existing solutions for FlexRay security and physical-layer key agreement.

A. FlexRay Security

The subject of FlexRay security has attracted less attention than security on CAN. One of the reasons may be that FlexRay is not directly exposed through the diagnostics port like CAN so it is less prone to adversarial interventions. One the other hand, the FlexRay protocol is more complex and dedicated development tools are less commonly available than those targeting CAN.

Still, a number of papers cover various aspects of security in FlexRay-based networks. Some lines of work were concerned with efficient scheduling of FlexRay communication in the presence of security-related transmissions. A security aware scheduling mechanism is presented in [10]. The authors consider joint scheduling of CPU tasks and FlexRay communication in the static segment to achieve message authentication using a TESLA-like scheme. Another more recent work that discusses optimized scheduling for the adoption of TESLA schemes for FlexRay can be found in [11]. The performance of using several broadcast authentication protocols on FlexRay and CAN-FD (CAN with Flexible Data-rate) is analysed in [12]. The employed protocols are also classified based on the key distribution model followed by each but the analysis is only done by simulation.

The problem of key distribution in FlexRay networks has minimal coverage in related works. In [13], the authors propose the use of pre-shared one-way chains as keys for MAC based authentication of FlexRay traffic. However, aspects related to key renewal and key storage memory requirements are not discussed. The authors of [14] discuss porting of a TESLA [15] based CAN authentication protocol to FlexRay. Their approach assumes that each network node has a set of hard-coded pre-shared keys that are employed in subsequent protocol steps. Another proposed approach is to use security aware communication controllers to provide device authentication and encrypted communication [16]. However, their approach is also based on pre-shared keys. A slightly different approach, described in [17], proposes the use of a central or master node which generates and distributes session keys to all other FlexRay nodes. The key distribution process is secured with one-way key chains pre-shared between the master and each of the other nodes.

The main disadvantages of the existing proposals on key management for FlexRay communication consist in the inability to refresh keys over the vehicle lifetime or the high memory requirements for storing one-way chains that ensure sufficient keys. The approach that we propose in this work alleviates these two problems as it provides an efficient and secure way to exchange keys whenever required without the need of storing long one-way chains. Also, while the key exchange mechanisms proposed so far rely on algorithms implemented at the application layer, our current approach is based on intrinsic properties of the FlexRay physical layer.

B. Physical Layer Key Agreement

Physical layer key agreement was proposed for both wired and wireless communication protocols. Mueller and Lothspeich [7] are the first to propose such an approach for CAN by making use of the intrinsic wired-AND behavior of the CAN physical layer. We use their scheme as a basis for our work by adapting it to the FlexRay specification. This main scheme is extended in [18] to support group key agreement for CAN. The authors later show that this approach is vulnerabile to physical layer probing attacks and provide several efficient solutions to alleviate these vulnerabilites [19]. The use of a similar principle of two devices transmitting simultaneously was proposed and implemented for Near Field Communication (NFC) in [20]. The basic approach for this type of NFC key agreement suffers from the same vulnerability to probing attacks as the CAN mechanism. Hence, the authors of [20] use a similar solution as proposed for the case of CAN by introducing randomness in physical signaling characteristics.

In wireless communication, physical layer key agreement, also known as channel-based key agreement, uses reciprocity of random fading wireless channels [21]. The principle is based on correlated wireless channel measurements that an eavesdropper cannot obtain due to its relative position to the two legitimate nodes. As presented in [21], implementing this mechanism in wireless networks also faces a series of challenges that need to be addressed.

III. The FlexRay Protocol

The introduction of advanced control and safety systems in cars came with increasing demands on data rate and reliability

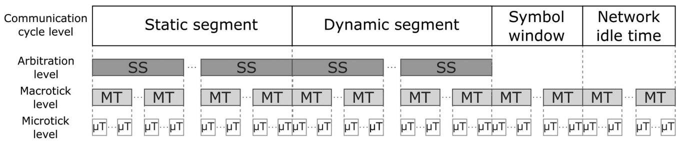

Fig. 2. The FlexRay communication cycle timing hierachy defined as sections that offer different levels of granularity. Following notations employed in the figure: static slot (SS), minislot (MS), macrotick(MT) and microtick (μT).

provided by in-vehicle communication channels. As a result, a new protocol was needed due to the inability of existing ones to fulfill these requirements. The solution came in the form of the FlexRay protocol developed to provide for deterministic and fault-tolerant communication. The last protocol specification is version 3.0.1 which was released in 2010 and has since been standardized in the form of ISO 17458. The core specification is separated in two main parts defining the data link layer [22] and the electrical physical layer [23].

FlexRay follows a time-triggered communication model based on TDMA (Time-Division Multiple Access). This means that all nodes follow a clearly defined communication schedule in which slots are assigned in each communication cycle for the transmission of specific messages. Fault-tolerance is assured through channel redundancy with two communication channels on each FlexRay node that can provide a data rate of up to 10 Mbit/s. If redundancy is not required, the two channels can be used in parallel to increase the data rate up to 20 Mbit/s. In terms of topology, FlexRay offers flexibility as it can be employed in any physical topology, e.g., bus, active/passive star or hybrid topologies.

The main body of the FlexRay protocol is described at the data link layer and is implemented by the FlexRay communication controller which can be implemented as a stand-alone chip or integrated as a microcontroller communication module. The electrical physical layer is implemented by the transceiver.

A. Communication Cycle

The communication schedule in a FlexRay cluster is fulfilled based on a periodical fixed length communication cycle which is structured into four main segments as depicted in Fig. 2: *static segment*, *dynamic segment*, *symbol window* and *network idle time*. Out of these only the static segment and the network idle time are mandatory in any FlexRay configuration while the remaining two can be added depending on communication requirements. Each segment has a predefined length which remains fixed in all communication cycles.

A static segment is comprised of a fixed number of *static slots* equal in length. Each static slot is statically allocated to a single node for the transmission of a frame with a given ID. A frame must always be transmitted in allocated static slots. Therefore, if the sender of a specific frame has no data available for transmission a frame with a ‘0’-padded payload, called a null frame, will be sent.

The dynamic segment was designed to support event-driven message transmission. The communication schedule must also pre-define the frames that could be transmitted in this segment.

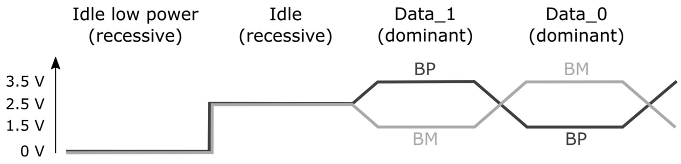

Fig. 3. The FlexRay physical bus levels.

However, in contrast with static segment rules, these frames will only be transmitted if required. To accommodate this behavior the dynamic segment is subdivided into smaller equally sized slices called *minislots*. A number of minislots is allocated for each frame that needs to be transmitted in the dynamic segment of a particular communication cycle until no further frames can be accommodated in the current cycle.

The symbol window is dedicated for simpler communication elements in the form of specific bit patterns. These bit patterns are called *symbols* and are used for tasks related to communication startup and network management. A network idle time segment is required at the end of any communication cycle to allow nodes to perform synchronization and communication cycle related tasks.

B. Time Representation

FlexRay employs a layered timing hierarchy to describe the communication cycle and achieve synchronization. The communication cycle, as depicted in Fig. 2, is composed of a fixed integer number of *macroticks*. All nodes in a FlexRay cluster have the same understanding of macrotick duration. Although some small variations are tolerated, greater mismatches in macrotick duration will lead to synchronization errors and can cause cluster-wide communication halt. A macrotick is represented as an integer number of microtics. Microticks are derived directly from local oscillator of each FlexRay node. Therefore, a distinct microtick duration may be employed by different network nodes which will also lead to a different number of microticks per macrotick.

An additional timing hierarchy level is used for the definition of the static and dynamic segments. The length of each of these segments is defined as an integer number of slots (static slots for the static segment and minislots for the dynamic segments). This level of abstraction is employed in the arbitration process which does not cover the symbol window and network idle time segments.

The nominal bit time is identical cluster-wide and defined as `gdBit` which can take one of three values depending on the employed bit rate. Hence, `gdBit` can be either 100, 200 or 400 ns corresponding to bit rates of 10, 5 and 2.5 Mbit/s respectively. `gdBit` is also derived from the local clock and is a multiple of microticks, i.e. either 4 or 8 microticks per `gdBit`.

C. Physical Layer Signaling

The physical layer of FlexRay is implemented using a two wire transmission medium for differential signaling. The two physical lines are denoted as BP (bus plus) and BM (bus minus). Fig. 3 presents the four physical line levels defined by the

TABLE I
OUTCOME OF FLEXRAY COLLISION TYPES

Node 1	*Data_0*	*Data_0*	*Data_0*	*Data_1*	*Data_1*	*Idle*
Node 2	*Data_0*	*Data_1*	*Idle*	*Data_1*	*Idle*	*Idle*
Bus signal	*Data_0*	Unpredictable	*Data_0*	*Data_1*	*Data_1*	*Idle*

TABLE II
BUS DRIVER TRANSMIT/RECEIVE BEHAVIOR

Transmit					Receive	
TxEN	**TxD**		**Bus signal**		**RxD**	**RxEN**
high	X	→	*Idle*	→	high	high
low	low	→	*Data_0*	→	low	low
low	high	→	*Data_1*	→	high	low

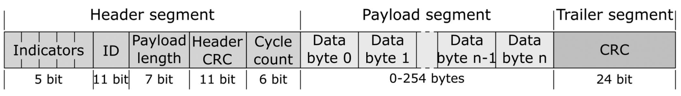

Fig. 4. The format of a FlexRay frame.

FlexRay electrical physical layer specification. Two of these levels are recessive while the other two are dominant. The two recessive levels are used to indicate a line in the idle state either during normal or low power mode. In contrast, the two dominant levels referred to as *Data_0* and *Data_1* are used to signal logical '0' and '1' respectively. FlexRay is designed for communication without collisions. Since these may happen during the startup phase, the specification provides a means to resolve some collision types. Table I lists the possible collision types and the resulting bus signal in each case. An important aspect on which we base our key agreement mechanism is that dominant signals (i.e. *Data_0* or *Data_1*) will always overwrite any recessive level (i.e. *Idle*).

For data transmission and reception the communication controller interfaces with the transceiver through four lines: TxD, TxEN, RxD and the optional RxEN. Therefore, there is a clear correlation between bus signals and the values produced on these lines as shown in Table II. The TxD and TxEN lines are used to control the signals generated on the bus while the RxD and RxEN are generated on the receive side according to existing bus signals. Table II illustrates the two transmit lines and resulting bus signal. While TxEN is set to high, the transceiver will output an *Idle* signal regardless of the value of TxD. The two dominant signals are generated by controlling the TxD line while keeping TxEN to low. On the receive side, the RxEN line indicates if the bus signal is recessive or dominant. However, this line is optional and absent in most of the FlexRay transceiver available on the market. For this reason we do not consider this line when building our key agreement scheme. The RxD transceiver line is high as long as the bus is *Idle* while in the dominant state it is set to low for *Data_0* and high for *Data_1* as indicated in Table II.

D. Frames and Symbols

The bulk of communication in a FlexRay cluster is achieved by transmitting frames in the static or dynamic segments. A FlexRay frame, depicted in Fig. 4, consists of three main segments: header, payload and trailer. A set of indicator bits at the begining of the header segment are used to specify information on the frame type and content. The frame identifier comes next followed by the payload length which indicates the number of 2 byte words contained in the payload field. The header also contains the counter of the current communication cycle and a cyclic redundancy code (CRC) computed over part of the indicator bits, ID and payload length. The payload field contains the transmitted data bytes while the trailer holds the CRC computed over the header and payload segments.

Besides frames FlexRay also uses simpler communication elements called symbols which take the form of specific bit patterns. FlexRay defines four symbols: collision avoidance symbol (CAS), media access test symbol (MTS), wakeup symbol (WUS) and wakeup during operation pattern (WUDOP). Only three patterns are defined to describe these symbols since CAS and MTS share the same pattern and can be distinguished by the communication state in which they are transmitted. CAS is used only for communication startup before the communication cycle is established while MTS is used during normal communication. Additionally, WUS is used for in cluster wakeup process before restarting normal communication. Therefore, only an MTS or WUDOP can be sent during a symbol window. The corresponding pattern for each symbol is illustrated in Fig. 5. As depicted by the figure, each symbol has several configurable features such as high/low/idle pulse duration or number of pattern repetitions. Pattern parametrization should be the same on all network nodes to allow for correct symbol decoding.

IV. SHARING SECRETS OVER FLEXRAY

In this section we describe the proposed approach for sharing secrets in a FlexRay cluster which relies on the inherent property of the physical layer that allows dominant levels to overwrite recessive ones. We begin by describing the underlying principle and then present how to employ it for key agreement in FlexRay clusters.

A. Basic Key Sharing Principle

To simplify the presentation we consider that a logical '1' is recessive while logical '0' is dominant and defer the discussion on the actual logical interpretation of recessive and dominant levels for the next section. In a similar fashion with the CAN protocol, the FlexRay physical layer implementation uses a wired-AND mechanism to resolve the bus level when dominant and recessive levels are generated at the same time by distinct nodes. Therefore, the resulting bus level will only be recessive if all nodes generate a recessive output, otherwise the resulting level will be dominant as long as at least one node generates a dominant output. This physical layer property was previously used to build a key agreement scheme for CAN [7]. We make use of the same underlying principle which we describe in what follows.

Let Alice and Bob be two network nodes that have to share a secret by taking advantage of the bus wired-AND behavior. Each of them independently generates a random bit sequence of predefined length n. They want to exploit the physical layer behavior by simultaneously transmitting their bit sequences in

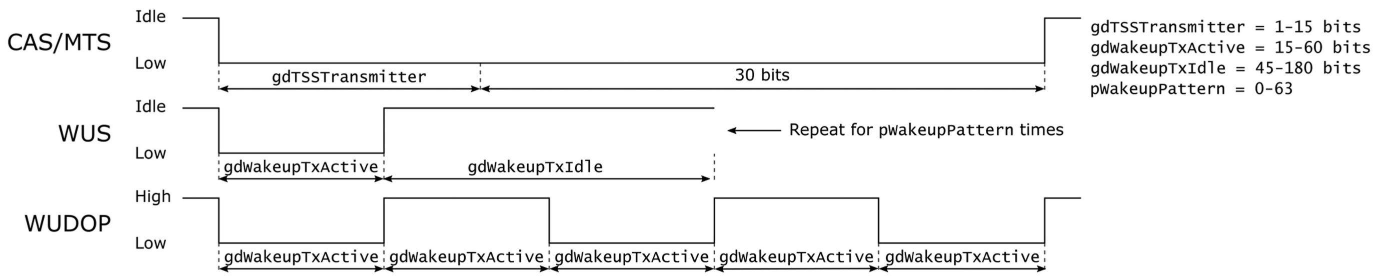

Fig. 5. FlexRay symbols: Collision Avoidance Symbol (CAS)/Media Access Test Symbol (MTS), Wakeup Symbol (WUS) and Wakeup During Operation Pattern (WUDOP).

a manner that would allow each of them to identify the value of the bit transmitted by its counterpart. Additionally, it should not be possible for an eavesdropper node Eve to determine the bit sequences sent by Alice and Bob by observing bus traffic. Clearly, for the information exchange to be successful it is also required that no other node, besides Alice and Bob, is transmitting during this interval. However, there are several issues in achieving these requirements. First, if both Alice and Bob transmit a '1' bit this will be evident to Eve since the resulting bus value is '1' only in this case. Secondly, if one of the nodes transmits a '0' it would be impossible for it to determine the value sent by its counterpart since the resulting bus value will be '0' regardless of the value transmitted by the second node.

According to the original scheme proposed by Mueller and Lothspeich [7] Alice and Bob overcome these difficulties by taking two steps. The first step, intended to enable each legit node to determine what its counterpart transmits, is to modify the generated random bit sequences by replacing each bit in the sequence by a bit pair containing the original bit followed by its inverse. These modified sequences are then transmitted as initially described. The second step is required to eliminate any leaked bits from the sequence. As a consequence of the first step, besides the case when both nodes transmit a '1,' Eve can now also determine when both nodes transmit a '0'. To solve this issue, after the bit transfer is complete Alice and Bob will drop all bits in their original sequence that correspond to a received bit pair containing a '1'. To better illustrate the previously described key sharing mechanism we give an example of operation for a 10 bit random input sequence in Fig. 6.

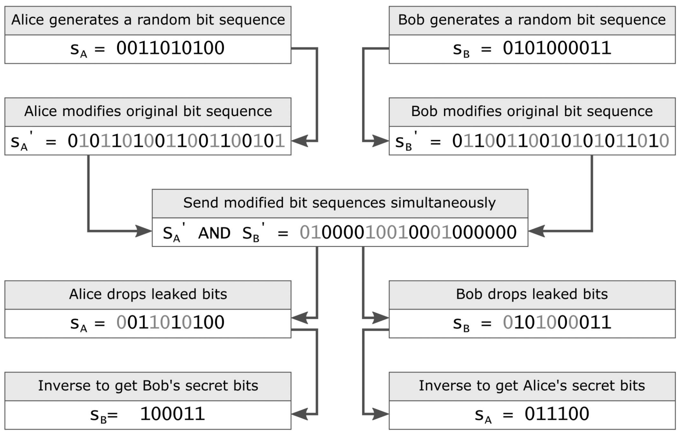
Alice generates a random bit sequence
s_A = 0011010100
Bob generates a random bit sequence
s_B = 0101000011
Alice modifies original bit sequence
s_A' = 01011010011001100101
Bob modifies original bit sequence
s_B' = 01100110010101011010
Send modified bit sequences simultaneously
s_A' AND s_B' = 01000001001000001000000
Alice drops leaked bits
s_A = 0011010100
Bob drops leaked bits
s_B = 0101000011
Inverse to get Bob's secret bits
s_B= 100011
Inverse to get Alice's secret bits
s_A = 011100

Fig. 6. Example of bus key sharing process for 10 bit random input sequences.

B. FlexRay Signaling for Secret Key Sharing

Implementing the secret key sharing mechanism described in the previous section on CAN is straightforward since the CAN physical layer encodes a recessive level as logical '1' and a dominant level as logical '0'. To implement the mechanism based on FlexRay communication the same wired-AND functionality must be provided. FlexRay uses the recessive level to denote an idle bus and the dominant level for one of the two active bus levels. As a transmitter, a node can switch between dominant and recessive output by means of the TxEN line (as shown in Table II). However, on the receive side a node cannot distinguish between the recessive state and the dominant Data_1 state unless its transceiver provides the RxEN line which is optional (first vs. third line in Table II). Moreover, according to the FlexRay physical layer specification [23] and Table I, the outcome of different network nodes generating distinct dominant levels at the same tame (i.e. Data_0 vs. Data_1) is undefined and depends on transceiver characteristics.

Even under these circumstances, a FlexRay node can still effectively generate recognizable dominant and recessive levels that can be used to implement the previously described key sharing scheme. To define a generic mechanism we assume that the RxEN line is not available. For key sharing, nodes shall consider a recessive *Idle* level as a logical '1' and a dominant *Data_0* level as a logical '0'. This can be translated into signaling at the communication controller interface lines (i.e. TxEN, TxD and RxD) according to Table II. Therefore, transmission can be implemented by always keeping the TxD line low and only toggling the TxEN line. Setting TxEN to low will result in a dominant bus level which is received as a low level on the receiver's RxD line. Similarly, a high level on the TxEN line results in a recessive bus level which is interpreted as a high level on the receiver side. This encoding effectively implements the desired wired-AND behavior as illustrated in Fig. 7 which shows the resulting RxD signal obtained when two FlexRay nodes toggle their TxEN lines. Both nodes hold a constant low level on the TxD line (TxD lines and actual bus signals and not shown to avoid cluttering the plot). The RxD output clearly

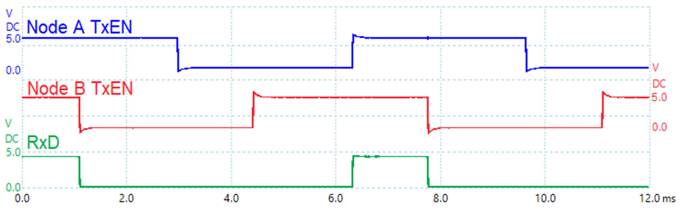

Fig. 7. RxD line levels of a FlexRay transceiver resulted from the toggling of the TxEN lines of two transmitter nodes which keep their TxD lines low.

provides the result of the AND operation applied to the TxEN input lines of the two sending transceivers.

C. Fitting Key Agreement in the FlexRay Communication Cycle

To assure backward compatibility with existing FlexRay protocol implementations (i.e., FlexRay controllers and transceivers), transmissions related to key agreement should be made in a manner that does not disrupt the normal operation of nodes that do not implement this mechanism. First off, since a key agreement service is used on a request basis it can only be accommodated by the dynamic segment or symbol window.

The dynamic segment relays event-triggered communication according to application-specific requirements. The communication load on the dynamic segment varies within a FlexRay cluster depending on the occurrence of message-triggering events and will also differ from one FlexRay cluster setup to another. Moreover, the communication controller expects correct frames to be transmitted within the dynamic segment or will otherwise report errors upon reception. However, using the FlexRay frame format to implement the proposed key exchange mechanism would not be possible since a correct frame is a sequence of Data_0 and Data_1 signals while the proposed key sharing mechanism generates a sequence of Idle and Data_0 signals.

Transmissions in the Symbol window are less frequent than transmissions in the dynamic segment. The MTS was defined to serve as a means of testing the proper functioning of the FlexRay *bus guardian* which in turn was intended to detect and confine errors generated by a network node. However, the FlexRay specification for the bus guardian is still in its preliminary form (since December 2005) [24] and, to the best of our knowledge, there exists no bus guardian implementation in use. Even if the bus guardian will be implemented in the future, the transmission of the MTS would be rare. WUDOP, the second symbol that can appear in a symbol window, is transmitted when the remote wakeup of a network node is required which also makes it an infrequent occurrence.

We propose to use the symbol window for the key sharing mechanism transmissions due to its usually sporadic use and the fact that symbols are not expected to have complex frame-like structures. FlexRay clusters functioning inside existing vehicles may not include the optional symbol window in their communication schedule. Retrofitting any existing clusters with the proposed key agreement mechanism can be done by firmware updates of network nodes. Therefore, the new firmware can

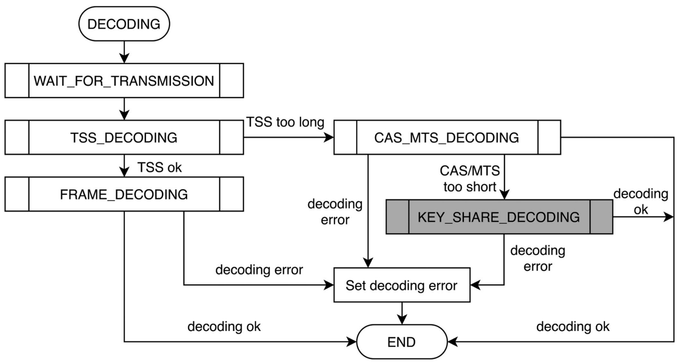

Fig. 8. Adaption of the FlexRay decoding procedure to include support for secret sharing (white background for original blocks and gray background for added block).

include an updated communication cycle which supports the symbol window. Communication cycle changes should be minimal to avoid affecting communication reliability (e.g. a small part of the dynamic segment could be reallocated to the symbol window). If symbol window inclusion in the communication cycle is not possible, the unused minislots at the end of the dynamic segment could be used as an alternative space for special transmissions required to implement key agreement. From the backward compatibility perspective, the effects of using this alternative approach are similar to the use of the symbol window as we show in Section VII-B.

D. Shared Key Sequence Encoding/Decoding

The existing FlexRay encoding/decoding mechanism must be considered when introducing an additional transmission format. This is required to allow correct decoding of the new transmission type and avoid collisions with existing frame and symbol formats. In Fig. 8 we depict the basic FlexRay transmission decoding process, which is applied in all communication segments, along with the changes required for full support of the new shared key sequence. Original processing blocks are presented as specified by the FlexRay standard [22] and marked with a white background while the required addition for key sequence decoding is marked with gray.

Upon detecting the start of a transmission the FlexRay decoding mechanism starts by interpreting the preamble known as the *transmission start sequence* (TSS) which consists of a series of bits set to *Data_0*. If the transmission is a frame it is expected that the number of bits in the TSS is equal to `gdTSSTransmitter`, a parameter specified when defining the FlexRay communication parameters. If the number of consecutive bits set to *Data_0* reaches `gdTSSTransmitter` $+ 2$ the transmission is considered to be a symbol and following bits are passed through the CAS/MTS symbol decoding block since this is the only symbol type that can be sent while the network is active. As shown in Fig. 5, for a successful decoding of the CAS/MTS symbol `gdTSSTransmitter` $+ 30$ consecutive *Data_0* bits have to be received otherwise a decoding error is reported. Note that the decoding error is only reported to the application layer and no

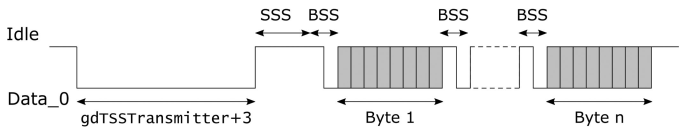

Fig. 9. Encoding of the key sharing sequence.

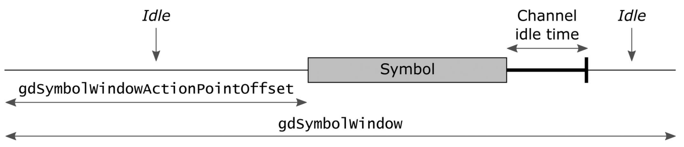

Fig. 10. Timing within the FlexRay symbol window.

automatic action is taken upon such an event as we discuss in the experimental section. With this in mind, we propose to set the TSS of the shared key bit sequence to $\texttt{gdTSSTransmitter} + 3$ followed by a transition to the *Idle state* to clearly differentiate it from a known frame or symbol transmission. This design choice also prevents unwanted collisions in case a MTS symbol is transmitted at the same time as the key sharing sequence. This is because the greater number of consecutive dominant bits in the MTS symbol will overwrite the recessive *Idle* bits from the key sharing sequence leading to postponing the key agreement procedure until the next cycle.

If the key agreement mechanism implementation is done at the FlexRay communication controller level, the additional KEY_SHARE_DECODING block should be included. This block will implement decoding of key bits along with the required arbitration mechanism to avoid collisions with other symbol transmissions. This eliminates unnecessary error reports caused by behavior undefined at the FlexRay communication controller level.

To define a format for encoding the shared key sequence transmission, we consider that length of the bit sequences is always a multiple of 8 bits so that it contains an integer number of bytes. Fig. 9 illustrates our proposal for encoding the bit sequence transmitted in the symbol window. The transmission starts with a TSS of $\texttt{gdTSSTransmitter} + 3$ bits set to *Data_0*. Next, a sequence of *Idle* bits indicates the start of the sharing sequence (SSS). We define a new global protocol parameter named `gdSecretSS` to specify the length of the SSS which should be at least 1 bit. Its upper bound can be calculated depending on the employed bit rate, length of the shared bit sequence and number of usable bits in the symbol window. The bytes in the sequence are transmitted next, each preceded by a byte start sequence (BSS) that consists of one *Idle* bit followed by a *Data_0* bit. The *Idle* level is re-established at the end of the transmission.

E. Shared Secret Sequence Length Considerations

Some implementation constraints have to be considered when calculating the maximum number of bits that can be transmitted in the symbol window. These constraints are mainly due to FlexRay specification parameters which we indicate as we enumerate these constraints in what follows:

- The symbol window can hold at most one symbol per communication cycle.
- The length of the symbol window, upper bounded by transmission medium characteristics, is specified by a global cluster parameter called `gdSymbolWindow` (MT).
- For clock synchronization purposes an idle time interval is required between the start of a segment and the transmission start within the segment. This interval is given as the global cluster parameter `gdSymbolWindowActionPointOffset` (MT).
- A channel idle time of at least 11 bits (`cChannelIdleDelimiter` bits) is required to signal the end of the symbol transmission.
- The format of the shared secret sequence, should be distinct from frames and other symbols in order to avoid collisions and correctly trigger the proper decoder element on the receiver side. As explained in Section IV-D, this is achieved by setting the length of the TSS preamble to $\texttt{gdTSSTransmitter} + 3$ bits.
- Since long sequences of *Idle* or *Data_0* bits can be generated, a mechanism is required to maintain synchronization and prevent misidentifying long *Idle* sequences as channel idle time (symbol end). We employ a bit stuffing mechanism which we discuss later in this section.

TABLE III
NUMBER OF USABLE BITS IN SYMBOL WINDOW

Bit rate (Mbit/s)	2.5	5	10
$\texttt{gdSymbolWindow}^{\text{Max}}$ (MT)	145	162	161
$\texttt{gdSymbolWindowActionPointOffset}^{\text{Max}}$ (MT)	63		
`cChannelIdleDelimiter` (bits)	11		
$\texttt{gdTSSTransmitter}^{\text{Max}}$ (bits)	5	8	15
`gdSecretSS` (bits)	1		
`gdBit/MT`	5		10
Number of usable bits (no stuffing)	**390**	**472**	**950***
Number of usable bits (CAN-like stuffing)	**347**	**420**	**845***
Number of usable bits (FlexRay-like stuffing)	**312**	**376**	**760***

* Actual usable values are half of those shown due to constraints discussed in the following Section IV-F.

Fig. 10 illustrates basic timing elements in a symbol window which limit the size of the symbol. By considering all previously mentioned restrictions (besides bit stuffing) the usable length of the symbol window can be calculated as $(\texttt{gdSymbolWindow} - \texttt{gdSymbolWindowActionPointOffset}) \times (\texttt{gdBit/MT}) - \texttt{cChannelIdleDelimiter} - (\texttt{gdTSSTransmitter} + 3) - \texttt{gdSecretSS}$.

The effective number of usable bits will differ depending on various cluster settings and the employed bit stuffing scheme. Table III illustrates the number of usable bit space considering the maximum allowed symbol window length and the three possible FlexRay bit rates. All other protocol parameters are also considered at their maximum values for determining a lower bound for usable bit space. Bit stuffing usage impact is illustrated by presenting results both for bit-stuffed and non bit-stuffed sequences. We examine two bit stuffing approaches. The first is inspired by CAN and consists of the insertion of an opposite value bit after each 8 consecutive bits of the same value. The second approach uses the same mechanism as the FlexRay frame encoding which is to precede each 8 bit sequence by a bit pair formed by an Idle bit followed by a Data_0 bit called BSS (byte start sequence). Thus, depending on the scheme, at most 1 or 2 bits are consumed for each 8 that have to be transmitted. While it needs fewer additional bits, the CAN-like bit stuffing is more difficult to implement as it requires nodes to monitor incoming bit sequences to decide on the polarity of the stuffing bit. The FlexRay approach is easier to implement and only requires nodes to expand the bit sequences with one BSS per byte. For the remainder of the paper we consider that FlexRay-like bit stuffing is used to augment the bit sequence of the shared key.

These calculations are made on a per-channel basis. Since each FlexRay node is equipped with two channels, the number of usable bits could be doubled by parallel transmissions of different bit sequences as long as redundancy is not required for the key sharing process. Also note that the actual space available for the original random bit sequences is half of the available usable bit space presented in Table III because of the need to send one inverted bit for each original one.

The size of the Action Point Offset (APO) parameter has a considerable influence on the number of available bits. Note that, when set to its maximum, the APO can span over more than one third of the symbol window. A different APO size can be configured for each transmission slot type (i.e. static, dynamic and symbol window) and represents the time between the start of a transmission slot and the actual start of the transmission within the slot. This offset is provided to accommodate imperfect synchronization caused by clock deviation between nodes and comes at the cost of reduction in bandwidth. While we use the maximum specified value of `gdSymbolWindowActionPointOffset` for establishing an upper bound on bandwidth reduction, values of APOs employed in FlexRay network configurations are smaller (e.g. 5 MT used for static slot APO in [25] while in [26] an APO of 3 MT is used for both static slots and minislots).

F. Bit-Rate During the Key Exchange

An important aspect to consider is the ability to sustain the configured bit rate for transmitting the secret shared sequence. Some implementations of the mechanism, especially those not employing a dedicated FlexRay controller supporting the key agreement mechanism, may not be capable of matching the configured FlexRay cluster bit rate. Since the symbol window is used for transmitting the proposed key agreement mechanism, the encoding of the shared sequence can be adapted to use a different bit duration than the one used in frame transmission as long as it is a multiple of the cluster bit rate. The bit duration to be used can be either pre-defined or established during the protocol initiation phase in which each node should transmit its maximum supported bit rate for key exchange. The two nodes will use the lowest of the two bit rates.

Using a lower bit rate for key exchange reduces the number of bits that can be transmitted in one communication cycle. An improvement can be made by not encoding fields in the preamble of the key exchange sequence (TSS and SSS fields) based on the lower bit rate. These fields can span for several bits and their duration is only restricted by their lower boundaries. Therefore, their length can be set to the smallest multiple of the shared sequence bit length which is greater than or equal to their lower boundary according to the cluster bit duration.

Limitation by specification. The timing requirements for the receiver-side detection of transitions between the Idle and active (Data_0 or Data_1) bus levels are different than the timings used for transitions between the two active bus levels. The FlexRay electrical physical layer specification [23] specifies a bus driver idle detection time (transitions from active to Idle) of 50–205 ns and an activity detection time (transitions from Idle to active) of 100–250 ns. These timings are longer than the maximum 22.5 ns required to detect transitions between Data_0 and Data_1. Hence, the bus driver specification introduces an inherent limitation in the bit rate achievable by our proposed mechanism as we show in the evaluation section. As a consequence, at 10 Mbit/s, the actual number of usable bits is half of the calculated values from Table III.

V. Key Sharing Protocol

We apply the process describing how secrets can be shared over FlexRay to build a key agreement protocol. This section presents the envisioned protocol and discusses its security in front of adversarial actions.

A. Protocol Description

Key-Exchange protocol. The proposed protocol for secure key agreement over FlexRay consists of the following set of actions that are run by any pair of *initiator-target* nodes (N_i, N_t):

1) $\mathsf{Req\text{-}KA}(N_i, N_t)$ in which the initiator of the key agreement N_i sends a request marked by a predefined constant reqKa to the target node N_t for starting the key agreement,
2) $\mathsf{Ack\text{-}KA}(N_t, N_i)$ in which the target node N_t acknowledges the acceptance of the key agreement request sent by the initiator by replying with a predefined constant reqAck,
3) $\mathsf{Gen}(\ell)$ in which each node generates $R_{id}, id \in \{i, t\}$, a sequence of $|R_{id}| = 4\ell$ random bits (where ℓ is a predefined constant)
   then each node expands his random bit sequence by inserting a complement of each bit from the original sequence either after each original bit or grouped at the end of the original sequence based on a common encoding format on both nodes, let the expanded sequence be $\widetilde{R}_{id}$ (note that $|\widetilde{R}_{id}| = 8\ell$),
4) $\mathsf{Exch}(\widetilde{R}_i, \widetilde{R}_t)$ in which the initiator and target nodes N_i, N_t transmit their extended bit sequences and read out the resulting value from the bus $\widetilde{R}^* = \widetilde{R}_i \ \wedge \ \widetilde{R}_t$, where $\wedge$ represents the bitwise AND operation,
5) $\mathsf{ExtractKey}(\widetilde{R}^*)$ in which the usable bits which remained secret after the Exch operation are counted and if they are less than $2\ell - \epsilon$ then the key agreement is restarted or else the common secret is obtained by passing the extracted bits from the bus trough some key derivation function $\mathcal{KD}$ (the key derivation function can be a simple keyed

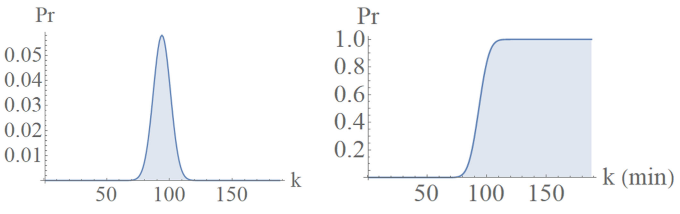

Fig. 11. Probability of extracting exactly k bits (left) and extracting less than k bits (right) for $\ell = 47, k \in [0..188]$.

hash-based message authentication code which uses the common shared secret as key and the message set to a predefined, publicly-known constant)

6) Ver in which nodes check the correctness of their version of the shared key by computing and sending the encryption of the default request message reqKa followed by the identity of the node or some other fixed constant for the initiator and target nodes.

In step ExtractKey($\widetilde{R}^*$) it can be easily seen that the initiator's extracted sequence is the exact inverse of the sequence obtained by the target node. Therefore the agreement initiator uses extracted bits as the shared secret key while the target node uses the inverse of its extracted usable bits so that the shared key is identical.

We requested that the number of generated bits is 4ℓ on each node, which later expands to 8ℓ, to comply with the stuffing procedure (which works at multiples of 8 bits). On average, half of the 4ℓ bits are lost which results in a session key that has an average size of 2ℓ bits. The probability of extracting exactly k bits out of the 4ℓ bits is given by the success rate of a random variable X describing a set of Bernoulli trials with $p = 0.5$, i.e., the probability of exactly k successes is:

$$\mathsf{Pr}[X = k] = \binom{4\ell}{k} 0.5^k 0.5^{4\ell - k}$$

The probability of extracting less than k bits can be computed as cumulative distribution function of random variable X, i.e.,

$$\mathsf{Pr}[X \le k] = \sum_{i=0,k} \binom{4\ell}{i} 0.5^i 0.5^{4\ell - i}$$

We now consider a more practical example based on the number of usable bits in the symbol window as presented in Table III. For a practical bitrate of 5 Mbit/s, there are 376 bits available which leads to $\ell = 47$ (the number of bits in the symbol window is divided by 8 according to step 3 of the protocol description). Fig. 11 shows the probability of extracting exactly k bits (left) and of extracting less than k bits (right) for $\ell = 47$ and $k \in [0..4\ell]$. On average, 94 bits can be extracted from each symbol window. Extracting 128 bits for an AES key may be set as reference point, a case in which 2 symbol windows will be needed. The probability of extracting less than 64 bits in one window is very small, i.e., 0.7×10^{-5}. On the other hand, the probability of extracting at least 64 bits in each window is greater than 0.9999 which is sufficiently high. In step 5 of the protocol we check that the number of extracted bits are at least $2\ell - \epsilon$ where ϵ is used as a tolerance margin (this bound is due to the fact that half of the 4ℓ bits are lost, i.e., leaked values). As a practical value, for $\ell = 47$ we can set $\epsilon = 30$ which results in keys that are at least 64 bits in length from each symbol window. The success probability of the key exchange, i.e., extracting at least 128 bits in two symbol windows, would be $1 - 3.03 \times 10^{-10}$ which is close to 1. In the highly unlikely event that less than 128 bits are extracted, the protocol will continue with a new symbol window until the expected number of bits are obtained.

B. Protocol Instantiation

We envision two main concepts for actual protocol implementation: one that does not involve the host in the key agreement process and a second one in which the host participates in the process. The first approach requires the communication controller to autonomously perform all steps of the key agreement and obtain the shared secret key. This key can be either provided to the host application to be used in host based security mechanisms or these operations can be implemented directly as part of the communication controller functionality to reduce host load. The approach of minimizing host load by implementing various communication-related tasks as custom extensions of the FlexRay communication controller was previously employed in by Shreejith *et al.* in [27]. The second concept assumes that the host application controls or performs all or some steps of the key agreement protocols.

While the actual key agreement mechanism was detailed in the previous section, we still have to account for the protocol initiation and key verification steps. The protocol is initiated by a request (reqKa) and response (reqAck) operation pair. This can be achieved by frame transmission in the dynamic segment since the key agreement operation is not performed often enough to justify slot allocation in the static slot. Moreover, both request and response can be transmitted in the same cycle by careful scheduling to allow the start of the actual key sharing operation in the nearest available symbol window. Several consecutive symbol window transmissions can be used if a single transmission does not provide enough entropy. We impose a limitation on the frequency of key agreement requests to restrict attacker capabilities. Therefore, a node should only accept a limited number of key agreement requests from another node in a short period of time. Exact values are to be defined according to application requirements. As a simple key verification mechanism we propose that each node uses a predefined symmetric key algorithm (e.g. AES) to encrypt verification information and sends it to its counterpart. The key is declared valid only if both nodes are successful in decrypting the verification message.

C. Attacker Model and Protocol Security

We consider two types of attackers that could target the proposed key agreement mechanism both having the same common goal of obtaining the key shared between two network nodes: eavesdropper and active attacker.

Eavesdropper: An eavesdropper is an attacker device that is plugged into the network with the ability to probe the physical network lines and extract timing or voltage characteristics of the signals without actively participating in communication. As discussed in related work on CAN [19], physical layer key agreement schemes are vulnerable to probing attacks. This is possible due to differences in propagation times and unique signal characteristics which may reveal the original random bit sequences sent by the two legit nodes. Fortunately, as proposed by the authors of the same paper [19], a series of countermeasures are available for this type of attacks. These consist in inducing slight random variations in the characteristics of each transmission to impede node identification.

Active attacker: The second type of adversary is an active attacker consisting in either a compromised or a plugged-in node. Such an attacker is capable of monitoring FlexRay traffic and generate transmissions on the communication line. While the scheme is secure in front of this type of adversary for passive topologies this is not true in an active star topology if the active star node behaves like a man in the middle. Such a man-in-the-middle attack is feasible since the FlexRay specification does not impose any rules on the implementation of communication forwarding between different active star branches. Similarly, if an adversarial node is able to physically interpose himself between two nodes in the network, the effect will be the same. The proposals from [7] are also vulnerable to such an adversary. Such attacks can be circumvented if the nodes that exchange the key have a previous shared secret. Then this secret can be used via a key-derivation function to authenticate the bits that are exchanged during the symbol window. An adversary which is not in possession of the key will fail to do so.

VI. Implementation

Using a modified communication controller would clearly be the more efficient approach from the computational point of view. Nevertheless, this will not only bring increased short-term costs, until new communication controllers are produced and made widely available, but will also delay integration in actual applications due to the time-to-market usually involved in new integrated circuit designs. For this reason we propose an approach that does not require the use of a communication controller by only employing the FlexRay transceiver directly interfaced with the microcontroller through general purpose I/O pins.

For our implementation we selected two automotive microcontroller families as representatives of different microcontroller classes. The employed microcontrollers are designed for FlexRay communication and come equipped with on-chip FlexRay controller modules. The first family is the NXP S12X representing the low to mid-end performance class, while the second is the Infineon TriCore AUDO family designed for high performance applications especially in the power train and chassis domains. Fig. 12 illustrates our experimental setup which consists of 2 S12X nodes and two TriCore AUDO FlexRay nodes.

A. NXP S12

1) Platform Description: Each of the employed EVB9S12XF512E development boards is equipped with a S12XF512 16-bit microcontroller from NXP and two TJA1080 FlexRay transceivers suitable for the implementation of a FlexRay node.

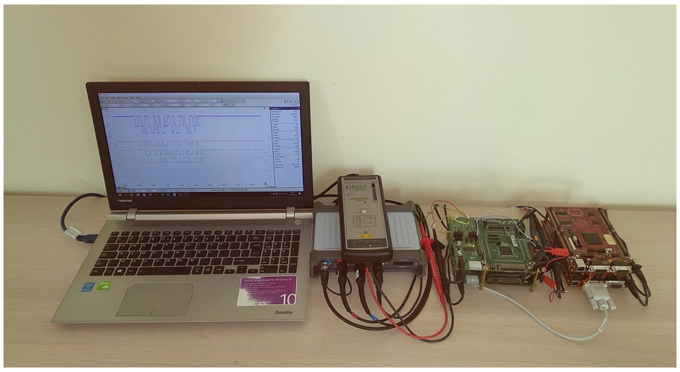

Fig. 12. Experimental setup comprised of EVB9S12XF512E and TriBoard development boards together with a PicoScope used for monitoring FlexRay signals.

The S12XF512 microcontroller is equipped with an XGATE co-processor, 512 KByte of Flash and 32 KByte of RAM offering a low to medium performance level among the automotive platforms. The XGATE co-processor is intended for improving performance by reducing the interrupt load of the main S12 CPU. Both the main CPU and the XGATE co-processor can be clocked at up to 100 MHz while other peripheral modules are provided clock signal with a frequency half that employed by the cores. The S12XF512 also features a FlexRay module capable of implementing communication at bit rates up to 10 Mbit/s.

2) Implementation Details: Given the limited performance of the S12 CPU an entirely software-based control of the FlexRay transceiver pins to implement the key agreement mechanism would not be feasible. Therefore, we make use of the on chip timer module for both encoding and decoding symbol window transmissions related to key sharing. Since each node has to simultaneously transmit and receive, we implemented the encoding on the main CPU while letting the XGATE co-processor in charge with decoding.

For transmitting the secretly shared sequence we use the timer in the output compare mode which toggles the associated channel pin once the timer counter reaches the configured value. The sequence to be transmitted is first transformed into an array representing time intervals between output level transitions. With the start of the sequence the time intervals in the array are programmed into the output compare register to generate the appropriate signal on the transceiver TxEN line.

For decoding the shared transmissions we employ another timer channel in the input capture mode which implements the opposite of the actions used for encoding. An input capture channel records the current value of the timer counter when triggered by a change of the associated pin state. We use this feature to buffer the time of all transitions of the Rx transceiver pin for the duration of the key exchange transmissions. Consecutive buffered values are then subtracted to obtain the time intervals between transitions. This information is further processed starting from the initial line level and the known bit rate to obtain the actual received bits.

B. Infineon TriCore AUDO

1) Platform Description: Two similar TriBoard development boards were used in our implementation each equipped with a different member of the Tricore AUDO family and different FlexRay transceiver circuits. The first is equipped with a TC1797 microcontroller and TJA1080 transceivers while the second uses a TC1782 microcontroller and AS8221 FlexRay transceivers. The two microcontrollers are built arround the same TriCore V1.3.1 core that can runt at up to 180Mhz. In terms of memory, the TC1782 offers 176 KByte of RAM and 2.5 MByte of Flash while the TC1797 is equipped with 156 KByte of RAM and 4 MByte of Flash.

2) Implementation Details: While the TriCore AUDO microcontrollers offer a considerably higher performance level compared to the S12X platform. Implementing the key exchange by only toggling the I/O pins from the application level is also not a viable solution for the TriCore when maximum transmission rates are required and when the CPU is also responsible for other tasks. Therefore we employed a similar approach as the one used on the S12X platform based on using the General Purpose Timer Array (GPTA) module.

The GPTA features a number of Local Timer Cells (LTC) intended for the generation or measurement of simple or complex digital signals. Each LTC can be used in one of four modes: freeruning timer, reset timer, capure and compare. Inter-cell communication capabilites makes it possible to implement complex functionalities such as coding and decoding for various communication protocols. One such example is the application note presenting the implementation of a receiver for the SENT protocol as described in [28].

We employed two groups of LTCs, one implementing the transmission and one for the reception. Cells in the transmission group were configured to function in compare mode. Cells toggle the configured transmit pin, one by one, at predefined time intervals which are pre-computed as based on the transmit sequence. Once half of the transmitting cells have performed their action they are reprogrammed with the next time intervals until the other half keeps generating output levels. The transmission stops when the entire sequence has been sent. For reception we employed a capture cell, connected to the receive pin, that triggers the DMA module to buffer the time elapsed between pin toggles. An additional compare cell is configured to implement a timeout functionality for detecting the transmission end. The buffered data is then processed as described in the S12X implementation.

VII. Evaluation

We evaluated our implementations both in terms of performance and compatibility with nodes not implementing the key agreement mechanism.

A. Achievable Key Agreement Bit Rate

1) NXP S12X: Aiming to achieve the highest possible bit rate for implementing the key exchange, we configured the maximum clock frequency for the timer module (50 Mhz in the case of the S12XF512). With this setting we were able to successfully encode and decode shared transmissions at a maximum bit rate of 1 Mbit/s using the S12XF215 platform. Fig. 13 illustrates the transmission of a shared sequence consisting of two bytes using 1 Mbit/s within a cluster communicating at 10 Mbit/s. Note that, in this case, the inverses of the generated random bits are sent separately (as bytes 3 and 4) instead of interleaving them with original bits. Node A is designated as the initiator and node B uses the TSS and SSS to synchronize before starting the transmission of its own sequence. The TSS (14 bits) and SSS (10 bits) sequences were sized to their closest representations at 1 Mbit/s (i.e. 2 and 1 bits respectively). As expected, the RxD line displays the result of applying the bitwise AND operation over the two receiving sequences with a slight delay caused by bus and transceivers propagation timings.

Our implementation on the S12XF512 platform is unable to sustain faster bit rates since operations required between pin toggles exceed the bit time. The required operations involve loading timer registers with the time interval to the next output level toggle or buffering the incoming toggle timings.

2) Infineon TriCore AUDO: Our TriCore implementation is capable of generating the digital secret bit sequences at higher bit rates compared to the S12X implementation. Bit rates of 10 Mbit/s (or even higher) can be obtained for generating the shared bit sequences on the digital outputs of the timer module. However, we were only able to obtain correct sequence transmissions at bit rates up to 5 Mbit/s, as illustrated in Fig. 14 for a test using the same bit sequences previously used in Fig. 13. This is caused by the time required by transceivers to detect transitions between Idle and active bus levels which is longer than the detection time of transitions between Data_0 and Data_1. The transmission of the secret sequences at bit rates higher than 5 Mbit/s is possible. However, the receiver circuitry in the transceiver is unable to detect levels with duration below the detection time thresholds. The TJA1080 specifies a time between 100 and 205 ns for both idle and activity detection while the AS8221 has an idle detection time of 50 to 200 ns and an activity detection time of 100 to 250 ns.

B. Backward Compatibility

We built a setup for evaluating backward compatibility of FlexRay traffic that includes key sharing sequences in the symbol window within a network with nodes that do not implement this mechanism. Two EVB9S12XF512E development boards were used as standard nodes to implement classic FlexRay communication at 10 Mbit/s. The communication cycle was configured to include all FlexRay segments (static, dynamic, symbol window and network idle time). The key sharing sequence was generated and sent from the TriBoard environment using a bit rate of 5 Mbit/s. We monitored the overall communication as well as the status returned by communication controllers from the S12XF microcontrollers for protocol error detection.

There are several use cases to consider for key sharing sequence transmission. First, we evaluated the case of sending the key sharing sequence in the symbol window segment without collisions with symbols sent by standard nodes, as illustrated

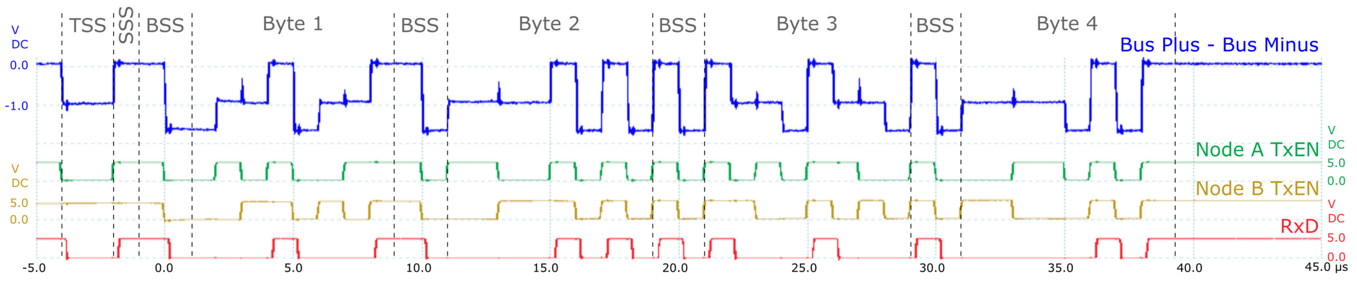

Fig. 13. Signaling during key exchange between the S12X nodes at 1 Mbps.

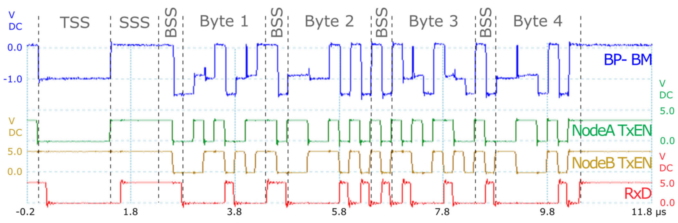

Fig. 14. Signaling during key exchange between the TriCore nodes at 5 Mbps.

TABLE IV
COMMUNICATION PARAMETERS AND CHARACTERISTICS OF THE BMW 7 SERIES FLEXRAY NETWORK [29]

Node count: 15	Bit rate: 10 Mbit/s	Cycle duration: 5 ms

Static segment		Dynamic segment	
Duration	3.003 ms	Duration	1.987 ms
Frame	16 bytes	Frame	2-254 bytes
Static slot count	91	Minislot count	289
Static slot duration	33 μs	Minislot duration	6.875 μs

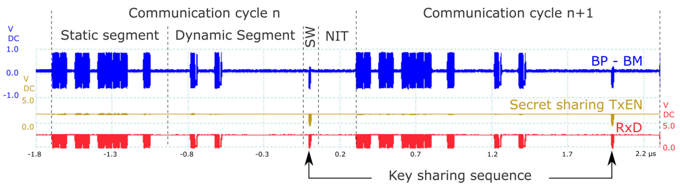

Fig. 15. FlexRay communication cycles including key sharing traffic.

in Fig. 15. This generates a symbol window syntax error on the receiving nodes since the key sharing sequence does not fit any of the specification defied symbols. This error is not used in the FlexRay automatic error confinement mechanism and does not generate message loss or communication halt. It is reported to the application layer where specific measures can be implemented. So the actual behavior of detecting such errors is application-specific.

A standard node sending one of the two predefined symbols allowed in the symbol window (MTS or WUDOP) while the key sharing operation is ongoing will lead to collisions and will also generate to a symbol window syntax error being reported by the communication controller. Additionally the key sharing operation will fail due to the unsuccessful execution of the protocol verification step. In such cases nodes implementing the key exchange can retry the operation after waiting for several cycles in which standard nodes can complete their symbol transmissions.

We also investigated the possibility of transmitting the key exchange sequence starting from the unused space at the end of the dynamic segment. Since correctly formed FlexRay frames are expected in this segment, the result is the communication controller reporting a syntax error for the minislots in the dynamic that contain the key sharing sequence. Like the syntax errors in the symbol window this error is also not used in the FlexRay error confinement mechanism. This error will not affect normal FlexRay communication unless any of the standard network nodes implements such a measure as an application layer mechanism in response to the reported error. The benefit of using empty minislots at the end of the dynamic segment is the possibility of sending longer sequences in one communication cycle beyond the bounds set by the maximum size of the symbol window.

C. Performance Analysis

We use the FlexRay network found in BMW 7 series vehicles [29] as an industrial use case to analyze the performance of the proposed FlexRay key agreement mechanism and to illustrate the impact of its introduction on the communication cycle.

The communication parameters of the FlexRay network designed for BMW 7 series vehicles are presented in Table IV. For the purpose of our analysis we consider that a portion of the communication cycle allocated to minislots in the dynamic segment is reallocated to provide space for a maximum sized symbol window. According to the parameters listed in Table III, for a 10 Mbit/s network, this would require reallocating a time space corresponding to approximately 33 minislots. This roughly corresponds to reducing the transmission capacity of the dynamic segment by three 20 byte frames or two frames with a payload of 56 bytes. The same assumption of the maximum APO (i.e. offset between the start of a transmission slot and the actual transmission time discussed in Section IV-E) was considered for the symbol window and the dynamic segment to provide an upper bound on protocol-related transmission overhead and a lower bound on transmission performance. A smaller APO reduces transmission overhead and enables higher transmission throughput in all communication cycle segments including the symbol window.

TABLE V
EXECUTION TIME OF ELLIPTIC CURVE OPERATIONS ON THE TRICORE PLATFORM

Algorithm	ECDH		ECDSA		ECDH		ECDSA	
Curve	SECP192R1				SECP256R1			
Key size	192				256			
Operation	Generate	Extract	Sign	Verify	Generate	Extract	Sign	Verify
Duration	84 ms	82 ms	84 ms	164 ms	136 ms	134 ms	142 ms	284 ms

TABLE VI
DURATION OF PHYSICAL LAYER AND ECDH KEY EXCHANGE ON THE TRICORE PLATFORM

	ECDH		**Our approach**		
	SECP192R1	SECP256R1			
	128 bit key		128 bit key	192 bit key	256 bit key
MAPO = 31 SWAPO = 63	414.24 ms	696.25 ms	5.107 ms	10.091 ms	10.123 ms
MAPO = 31 SWAPO = 31	414.24 ms	696.25 ms	5.047 ms	5.079 ms	10.047 ms
MAPO = 20 SWAPO = 20	414.21 ms	696.22 ms	0.095 ms	5.058 ms	5.090 ms
MAPO = 10 SWAPO = 10	414.18 ms	696.19 ms	0.081 ms	5.039 ms	5.071 ms
MAPO = 5 SWAPO = 5	414.16 ms	696.18 ms	0.073 ms	5.030 ms	5.062 ms

MAPO - gdMinislotActionPointOffset
SWAPO - gdSymbolWindowActionPointOffset

We now compare the performance of our approach to a classical key-exchange based on the widely used Diffie-Hellman protocol [30] implemented on a FlexRay network. We opt for an implementation based on elliptic curves to minimize the communication overhead. Table V shows the time required to perform the basic computational operations of the elliptic curve Diffie-Hellman key exchange (ECDH) and the elliptical curve DSA (ECDSA) as measured on the TriCore platform using implementations from the WolfCrypt [31] library. The signature is required in order to authenticate the Diffie-Hellman key exchange which is otherwise vulnerable to a man-in-the-middle attack. We were unable to run the same operations on the S12X platform due to limited amount of available RAM. This failure to port expensive public-key operations on low-end controllers is also an argument in favour to the proposed solution.

Table VI illustrates the time required to perform the key exchange operation, on 192 and 256 bit curves, then compares these to the time required for extracting a key of 128, 192 and 256 bits using the proposed approach. The 128 bit security level is selected as a baseline for our protocol since a 256 bit curve offers a security level that is equivalent to a 128 bit symmetric key, according to NIST recommendations [32]. We also keep the larger 192 and 256 bit keys to provide hints on scalability over distinct key sizes. The duration covers both the computational and communication overheads. The transmissions required for the Diffie-Hellman key exchange will be done in the dynamic segment and we assume that the key exchange messages are sent within the same communication cycle. This is indeed feasible since the Diffie-Hellman key shares will require one point of the elliptic curve, i.e. 192 or 256 bits plus one bit for the sign if compressed coordinates are used, while the DSA signature will be 320 bits in its smallest instantiation. The first row of the table considers maximum values of APOs for minislots and symbol window to establish a lower bound for performance. Following rows correspond to shorter APOs to illustrate performance changes due to improvement of communication throughput. In the case of ECDH, the duration of the key agreement mostly consist of the computational time required to perform elliptic curve operations. The time spent transmitting the frames required by the protocol is less than 200 μs for both key sizes. The durations listed for the physical layer approach are obtained by considering that, on average, half of the bits in the initially generated random sequence are leaked bits. Therefore, if one cycle is not enough to generate sufficient key bits the process is continued in the following cycles to obtain remaining bits. For the current proposal, the computational overhead is insignificant in comparison to the communication overhead. Reducing APO values brings improvements in both approaches, however, these improvements are less significant in the ECDH case. Overall, the physical layer based approach stands out as being considerably faster than the ECDH.

VIII. CONCLUSION

Our work proposes an approach for achieving key sharing in a FlexRay cluster by taking advantage of the physical layer characteristics. We demonstrate its feasibility by implementing it at the application layer on two different automotive platforms. The proposed key sharing mechanism can be integrated in existing FlexRay clusters and is compatible with nodes unaware of it as long as these nodes do not implement strict measures for handling syntax errors. The main advantage of the proposed mechanism is that it is transparent for other nodes on the bus and it takes advantage of the less utilized space in the symbol window. As future work we consider implementing the functionality as part of a FlexRay communication controller to improve performance and alleviate the problem of reported syntax errors during the key sharing operation.

ACKNOWLEDGMENT

The authors thank Prof. Dorina Isar and Sorin Popescu for their help in solving electronical problems in the experimental setup.

REFERENCES

[1] M. Wolf, A. Weimerskirch, and C. Paar, "Security in automotive bus systems," in *Proc. Workshop Embedded Secur. Cars*, 2004, pp. 1–13.
[2] D. K. Nilsson, U. E. Larson, F. Picasso, and E. Jonsson, "A first simulation of attacks in the automotive network communications protocol FlexRay," in *Proc. Int. Workshop Comput. Intell. Secur. Inf. Syst.*, 2009, pp. 84–91.
[3] P.-S. Murvay and B. Groza, "Practical security exploits of the FlexRay in-vehicle communication protocol," in *Proc. 13th Int. Conf. Risks Secur. Internet Syst.*, 2018, pp. 172–187.
[4] J. Van den Herrewegen and F. D. Garcia, "Beneath the bonnet: A breakdown of diagnostic security," in *Proc. Eur. Symp. Res. Comput. Secur.*, 2018, pp. 305–324.
[5] S. Checkoway *et al.*, "Comprehensive experimental analyses of automotive attack surfaces," in *Proc. USENIX Secur. Symp.*, 2011, pp. 77–92.
[6] C. Miller and C. Valasek, "Remote exploitation of an unaltered passenger vehicle," *Black Hat USA*, vol. 2015, pp. 1–91, 2015.
[7] A. Mueller and T. Lothspeich, "Plug-and-secure communication for CAN," *CAN Newslett.*, vol. 4, pp. 10–14, 2015.
[8] *Specification of Secure Onboard Communication*, 4th ed., AUTOSAR, 2017.

[9] *Specification of Crypto Abstraction Library*, 4th ed., AUTOSAR, 2015.
[10] Z. Gu, G. Han, H. Zeng, and Q. Zhao, "Security-aware mapping and scheduling with hardware co-processors for FlexRay-based distributed embedded systems," *IEEE Trans. Parallel Distrib. Syst.*, vol. 27, no. 10, pp. 3044–3057, Oct. 2016.
[11] R. Zhao, G. H. Qin, H. P. Chen, J. Qin, and J. Yan, "Security-aware scheduling for FlexRay-based real-time automotive systems," *Math. Problems Eng.*, vol. 2019, pp. 1–17, 2019.
[12] P. Vasile, B. Groza, and S. Murvay, "Performance analysis of broadcast authentication protocols on CAN-FD and FlexRay," in *Proc. WESS'15: Workshop Embedded Syst. Secur.*, 2015, pp. 7:1–7:8.
[13] D. Püllen, N. A. Anagnostopoulos, T. Arul, and S. Katzenbeisser, "Security and safety co-engineering of the FlexRay bus in vehicular networks," in *Proc. Int. Conf. Omni-Layer Intell. Syst.*, 2019, pp. 31–37.
[14] A. R. Mousa, P. Noureldeen, M. Azer, and M. Allam, "Lightweight authentication protocol deployment over FlexRay," in *Proc. 10th Int. Conf. Informat. Syst.*, 2016, pp. 233–239.
[15] A. Perrig, R. Canetti, J. D. Tygar, and D. Song, "The TESLA broadcast authentication protocol," *Rsa Cryptobytes*, vol. 5, pp. 2–13, 2005.
[16] S. Shreejith and S. A. Fahmy, "Security aware network controllers for next generation automotive embedded systems," in *Proc. 52nd Annu. Des. Autom. Conf.*, 2015, pp. 1–6.
[17] P.-S. Murvay, L. Popa, and B. Groza, "Accommodating time-triggered authentication to FlexRay demands," in *Proc. 3rd Central Eur. Cybersecurity Conf.*, 2019, pp. 1–6.
[18] S. Jain and J. Guajardo, "Physical layer group key agreement for automotive controller area networks," in *Proc. Int. Conf. Cryptographic Hardw. Embedded Syst.*, 2016, pp. 85–105.
[19] S. Jain, Q. Wang, M. T. Arafin, and J. Guajardo, "Probing attacks on physical layer key agreement for automotive controller area networks (extended version)," 2018, *arXiv:1810.07305*.
[20] R. Jin, X. Du, Z. Deng, K. Zeng, and J. Xu, "Practical secret key agreement for full-duplex near field communications," *IEEE Trans. Mobile Comput.*, vol. 15, no. 4, pp. 938–951, Apr. 2016.
[21] K. Zeng, "Physical layer key generation in wireless networks: Challenges and opportunities," *IEEE Commun. Mag.*, vol. 53, no. 6, pp. 33–39, Jun. 2015.
[22] ISO 17458-2, "Road vehicles – FlexRay communications system – Part 2: Data link layer specification," Standard, 2013.
[23] ISO 17458-4, "Road vehicles – FlexRay communications system – Part 4: Electrical physical layer specification," Standard, 2013.
[24] FlexRay Consortium, *Preliminary Node-Local Bus Guardian Specification, Version 2.0.9*, Dec. 15, 2015.
[25] P. Milbredt, A. Steininger, and M. Horauer, "Automated testing of FlexRay clusters for system inconsistencies in automotive networks," in *Proc. 4th IEEE Int. Symp. Electron. Des., Test Appl.*, 2008, pp. 533–538.
[26] E. Armengaud *et al.*, "Combining the advantages of simulation and prototyping for the validation of dependable communication architectures: the teodacs approach," *SAE Int. J. Passenger Cars-Electron. Elect. Syst.*, vol. 2, no. 2009-01-0763, pp. 309–318, 2009.
[27] S. Shreejith and S. A. Fahmy, "Extensible FlexRay communication controller for FPGA-based automotive systems," *IEEE Trans. Veh. Technol.*, vol. 64, no. 2, pp. 453–465, Feb. 2015.
[28] Alann Denais, *AP32123: TC1796 SENT Receiver (CPU & PCP implementation), Version V1.2*, Infineon Technologies AG, Aug. 2008.
[29] J. Berwanger, M. Peteratzinger, and A. Schedl, "FlexRay startet durch - FlexRay-bordnetz für fahrdynamik und fahrerassistenzsysteme (in german)," [Online]. Available: https://www.elektroniknet.de/flexray-startet-durch-1127.html, Nov. 2008, Accessed on: Feb. 20, 2020.
[30] W. Diffie and M. Hellman, "New directions in cryptography," *IEEE Trans. Inf. Theor.*, vol. 22, no. 6, pp. 644–654, Sep. 1976.
[31] wolfSSL Inc., "wolfCrypt embedded crypto engine," [Online]. Available: https://www.wolfssl.com/products/wolfcrypt/, Accessed on: Feb. 20, 2020.
[32] E. Barker, "NIST special publication 800-57 part 1, revision 4," NIST, Gaithersburg, MD, USA, Tech. Rep 800-57, 2016.

Pal-Stefan Murvay received the B.Sc. and M.Sc., and Ph.D. degrees University of Timisoara (UPT), in 2008, 2010, and 2014, respectively. He is a Lecturer at UPT. He has a 10-year background as a Software Developer in the automotive industry. He is currently leading the SEVEN project related to automotive and industrial systems security funded by the Romanian National Authority for Scientific Research and Innovation. He worked as a Postdoctoral Researcher in the CSEAMAN project and is currently a Senior Researcher in the PRESENCE project. His current research interests are in the area of automotive and industrial systems security.

Bogdan Groza received the Dipl.Ing. and Ph.D. degrees from University of Timisoara (UPT) in 2004 and 2008 respectively. In 2016 he successfully defended his habilitation thesis having as core subject the design of cryptographic security for automotive embedded devices and networks. He is a Professor at UPT, where he has been actively involved with the development of laboratories by Continental Automotive and Vector Informatik. Besides regular participation in national and international research projects in information security, he lead the CSEAMAN Project (2015–2017) and currently leads the PRESENCE Project (2018–2019), two research programs dedicated to automotive security funded by the Romanian National Authority for Scientific Research and Innovation.