Supporting Security Protocols on CAN-Based Networks

Gedare Bloom*, Gianluca Cena†, Ivan Cibrario Bertolotti†, Tingting Hu‡, Adriano Valenzano†
*Howard University, 2300 6th St NW, Washington, DC 20059, USA
Email: gedare@scs.howard.edu
†CNR–IEIIT, c.so Duca degli Abruzzi 24, I-10129 Torino, Italy
Email: {gianluca.cena, ivan.cibrario, adriano.valenzano}@ieiit.cnr.it
‡University of Luxembourg–FSTC, 6 rue Richard Coudenhove-Kalergi, L-1359 Luxembourg
Email: tingting.hu@uni.lu

***Abstract*—The ever-increasing variety of services built on top of the Controller Area Network (CAN), along with the recent discovery of vulnerabilities in CAN-based automotive systems (some of them demonstrated in practice), stimulated a renewed attention to security-oriented enhancements of the CAN protocol. The issue is further compounded nowadays because, unlike in the past, security can no longer be enforced by physical bus segregation.**

This paper describes how CAN XR, a recently proposed extension of the CAN data-link layer, can effectively support the distributed calculation of arbitrary binary Boolean functions, which are the foundation of most security protocols, without necessarily disclosing their operands on the bus. The feasibility of the approach is then shown through experimental evaluation and by confirming its applicability to a shared key generation protocol proposed in literature.

***Index Terms*—Network security, Cryptographic protocols, Controller area network (CAN).**

I. Motivation and Related Work

Although the Controller Area Network (CAN) [1] was conceived primarily as a real-time bus for engine-related control functions at its inception [2], it is nowadays used for an ever-increasing variety of applications and services. For instance, it gained popularity in on-board vehicle diagnostics [3] and industrial automation [4].

Starting in 2010 [5] researchers also gathered compelling evidence that CAN security features were fairly weak, eventually leading to practically demonstrated attacks [6]. The issue is made even more complex by the fact that it is no longer possible to enforce "security by obscurity" or resort to physical bus segregation, like it was done in the past [7]. In fact, infotainment equipment connected to critical on-board CAN buses was shown to be a viable attack target [8], even through a wireless channel.

As a result, there is a strong need to enhance the CAN protocol data-link layer to better support security-oriented protocols. Those enhancements ought to satisfy two key features, namely low overhead (given the very limited maximum CAN payload size) and compatibility with standard CAN controllers.

CAN with eXtensible in-frame Reply (CAN XR) [9] is a recent proposal that brings a number of enhancements to the CAN data-link layer. The basic idea behind CAN XR is to let more than one node transmit concurrently on the bus during the payload transmission phase (hence not only in the arbitration phase), while maintaining backward compatibility with legacy CAN controllers. By means of this feature, as it will be demonstrated in the following, it is possible to efficiently calculate any binary Boolean function directly on the bus without necessarily disclosing its operands to bus observers.

To confirm applicability to protocols of practical interest, we focus on previous work [10], in which the authors propose a key establishment protocol between two CAN nodes. In particular, as it will be shown in this paper, the proposed protocol is based on performing an Exclusive Nor (XNOR) operation, which is just a special case of binary Boolean function and can be conveniently carried out on the bus by means of CAN XR. Using CAN XR as a framework also addresses most open issues pointed out in [10], for instance, proper handling of bit stuffing and of the Cyclic Redundancy Check (CRC) field.

Moreover, this approach also ensures correct synchronization between nodes that send overlapping data on the bus. An additional shortcoming mentioned in [10]—namely, insufficient payload length—can easily be addressed by means of fragmentation and reassembly mechanisms that have already been applied to other CAN-based protocols [11], [12] and can coexist with CAN XR.

The paper is organized as follows. Section II summarizes CAN XR and extends [9] by providing a rigorous definition of its behavior upon multiple, overlapping bus transmissions. Then, Section III shows how to leverage CAN XR primitives to calculate arbitrary binary Boolean functions and Section IV redefines the protocol proposed in [10] in terms of a distributed XNOR operation. Section V presents experimental results obtained from a prototype implementation and Section VI concludes the paper.

II. CAN XR and Overlapping Transmissions

A. Protocol Definition

As outlined previously, CAN XR allows multiple nodes to transmit concurrently within the data field of the same bus *transaction*, rather than only during arbitration, as standard CAN does. Although CAN XR requires a new breed of

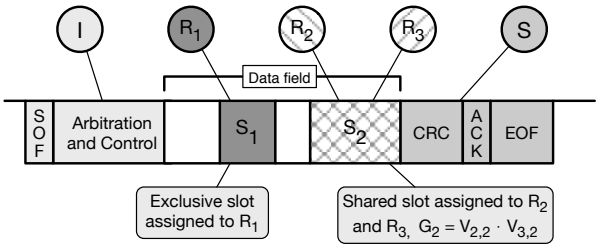

Fig. 1. Example of relationship between I, S, R_i, and S_j.

TABLE I
NOTATION (MESSAGE IDENTIFIER LEFT IMPLICIT)

Symbol	Meaning
I	Initiator
S	Supervisor
R_i	Responders
C_i	Consumers
F_i	Followers (either responders or consumers)
S_j	Slot within the data field
$V_{i,j}$	Value sent by R_i in S_j
G_j	Bus value in S_j

controllers to be properly implemented, the resulting bit stream on the bus is indistinguishable from a standard CAN frame exchange. In this way, CAN XR nodes can coexist with existing protocols and devices.

In any given XR transaction, which is uniquely identified by means of the CAN message identifier, nodes can assume zero or more *roles*, described in the following.

1) The *initiator* I initiates the transaction by sending the SOF, as well as the arbitration and control fields, up to and including the DLC, which sets the data field length.
2) A number of *responders* R_i inject on the bus their *replies* at appropriate places (called *slots* in the following) within the data field, while also receiving the frame.
3) *Consumers* C_i, which can be ordinary CAN nodes, just receive the frame. Responders and consumers are collectively called *followers* and denoted by F_i.
4) The *supervisor* S (typically coinciding with I) inserts stuff bits where needed, computes and sends the CRC, manages the ACK slot, and deals with EOF and intermission.

All nodes except S acknowledge the frame upon successfully receiving it, as ordinary CAN nodes do. Each R_i is responsible for inserting stuff bits while it is transmitting on the bus. S does the same for the whole frame, thus ensuring that bus traffic complies with the CAN standard even though one or more R_i may go missing.

B. Data Slots

As depicted in Fig. 1, the data field of any given bus transaction is divided into a number of slots S_j. Each F_i is configured to either write into or read from a subset of those slots, all the others being implicitly ignored. An F_i for which no write slots have been set becomes a C_i for the transaction, the others are R_i.

Generally speaking, slots can be either *static* and located in predefined, configured positions of the data field depending on the transaction's CAN identifier, or *dynamic*, in which case they are managed according to a minislotting approach (linear arbitration) [9], [13]. Regardless of how slot S_j is located, two cases are possible depending on how many R_i are configured to write into it.

1) If there is exactly one R_i, the slot is reserved to its own, *exclusive* use.
2) If there is more than one R_i, the slot is *shared* among them, and their replies will overlap.

In the following, we focus only on static, shared slots. At the CAN physical layer there are two complementary bus states called *recessive* (1) and *dominant* (0). Regardless of the physical layer implementation, in the case of simultaneous transmission of recessive and dominant bits by multiple nodes, the latter prevails on the former and the resulting bus state is dominant [1, Clause 4.6].

It is therefore straightforward to prove that, if a number of responders R_i send simultaneously a bit value V_i on the bus, the resulting bus level is their Boolean product (AND) $G = \prod_i V_i$. By extension, the same is true also for (multi-bit) shared slots, as illustrated in Fig. 1. If R_i is a responder in S_j, denoting by $V_{i,j}$ the value sent by R_i in S_j, and by G_j the resulting bus value, then $G_j = \prod_i V_{i,j}$.

It is also worthwhile to notice that this property of the CAN bus is in no way a borderline case. In fact, the standard CAN protocol relies on it to implement content-based arbitration [1, Clause 4.10], a key protocol mechanism. Table I summarizes the notation introduced so far.

III. Binary Boolean Functions in CAN XR

It is well known that there are 2^{2^n} possible Boolean functions $f : \mathcal{D}^n \to \mathcal{D}$, where $\mathcal{D}$ is a Boolean domain defined as $\mathcal{D} = \{0, 1\}$ in the following, and $n \geq 1$ is the arity of the function.

In particular, there are 16 possible Boolean functions $f_c(A, B)$ of two operands $(A, B) \in \mathcal{D}^2$, that is, binary Boolean functions for which $n = 2$. The index $0 \leq c < 16$ uniquely identifies one of them. It is possible to express $f_c(A, B)$ as a weighted sum of *minterms*

$$f_c(A, B) = c_0 \, \overline{A} \, \overline{B} + c_1 \, A \, \overline{B} + c_2 \, \overline{A} \, B + c_3 \, A \, B, \quad (1)$$

where $\overline{A}$ denotes the complement of A (NOT), arithmetic operators denote Boolean product and sum (AND and OR), and c_i are the digits of the binary representation of c.

In the following, we show that it is possible to calculate $f_c(A, B)$ for any c in a distributed way, when A and B reside on two different CAN XR responders R_A and R_B, without necessarily disclosing A and B to bus observers. The calculation is carried out by means of a single frame exchange and a cluster of $k \leq 4$ bits of a static, shared slot. The only underlying assumption is that c is known to both nodes. The

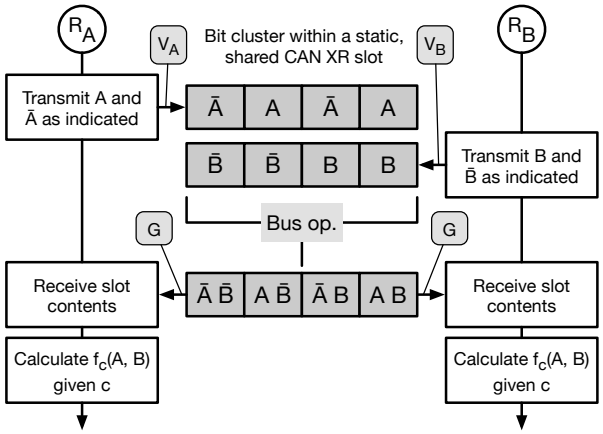

Fig. 2. Binary Boolean function calculation (not optimized).

method can be extended to handle the case in which A and B are multi-digit binary values and f is a bit-by-bit binary Boolean function, by simply using a k-bit cluster for each bit.

The idea is to split Equation (1) into three parts. Two of them can be calculated by each node in isolation, without knowing the value held by the other node, and the third one is calculated by the *bus* itself during data transfer. In particular:

1) both R_A and R_B calculate the complement of their value, $\overline{A}$ and $\overline{B}$ respectively; then, they build the 4-bit strings $V_A = \overline{A}, A, \overline{A}, A$ and $V_B = \overline{B}, \overline{B}, B, B$ to be sent on the bus;
2) during the frame exchange, the bus calculates the Boolean product $G = V_A V_B$ as described in Section II; denoting by $G^{(k)}$ the k-th bit of G, this corresponds to the 4 minterms $G^{(0)} = \overline{A}\,\overline{B}, \ldots, G^{(3)} = A\,B$;
3) given the $G^{(k)}$ received from the bus, both A and B calculate $f_c(A, B) = \sum_{k=0}^{3} c_k G^{(k)}$.

Fig. 2 illustrates the algorithm and the frame exchange in more detail. As shown in the figure, R_A and R_B transmit their value and its complement twice within a 4-bit cluster of a fully overlapping CAN XR slot, as indicated. Then, they gather the result of the wired-AND bus operation and calculate $f_c(A, B)$.

Although the worst-case cluster length is $k = 4$, if c is fixed and known in advance then one can ensure a cluster length of $k \leq 2$ by leveraging two observations. First is that any f_c having more than two bits in c *set* (namely $c_i = 1$) can be substituted by $f_c = \overline{f_{\overline{c}}}$ where $f_{\overline{c}}$ is the Boolean dual of f_c and, in our formulation, is exactly the function obtained by using $\overline{c}$ in place of c. This first observation is a direct application of Boolean duality under de Morgan's laws. The second observation is that clusters may be shortened for efficiency by omitting terms in which $c_i = 0$ (respectively, $\overline{c_i}$ in case the dual is in use).

As an example, we note that the XOR function, denoted by $\oplus$, and its complement XNOR are defined as

$$A \oplus B = \overline{A}\,B + A\,\overline{B}, \text{ and } \overline{A \oplus B} = \overline{A}\,\overline{B} + A\,B. \quad (2)$$

TABLE II
XNOR OPERATION ON THE CAN BUS

Operands				Bus bits		Result	Key bit
A	B	$\overline{A}$	$\overline{B}$	$A\,B$	$\overline{A}\,\overline{B}$	$X = \overline{A \oplus B}$	K
0	0	1	1	0	1	1	—
0	1	1	0	0	0	0	0
1	0	0	1	0	0	0	1
1	1	0	0	1	0	1	—

Hence, as already proved in general terms, $k = 2$ suffices to calculate XOR (respectively, XNOR), which is of particular interest for the discussion in Section IV.

For what concerns operand disclosure, it is worth mentioning that neither A nor B can be observed directly on the bus. Then, the amount of information a (possibly malevolent) bus observer can gather depends on which minterms are transmitted and their value. For instance, if the minterm AB is transmitted on the bus and its value is 1, the bus observer can infer that $A = B = 1$ for certain. On the other hand, if it is 0, the observer gains more limited information and cannot distinguish between the three remaining combinations of values of A and B.

IV. XNOR-Based Key Establishment

The goal of the protocol defined in [10] is to establish a shared secret between two nodes R_A and R_B. In the following we initially consider, for simplicity, a 1-bit secret.

A. Protocol Reformulation

After being reformulated according to the concepts presented in Sections II and III, the protocol works as follows.

1) Both R_A and R_B generate a 1-bit random value, A and B, respectively. All possible combinations are listed in the four leftmost columns of Table II.
2) They calculate the XNOR between A and B, that is, $X = \overline{A \oplus B}$, according to (2) and using an optimized version of the procedure depicted in Figure 2, based upon a *2-bit* cluster consisting of the first and last element of the 4-bit cluster shown in the figure. Cluster contents are shown in the fifth and sixth column of the table. The seventh column lists the corresponding value of X.

Depending on the value of X, two cases are possible:

3) If $X = 1$, then $A = B$ and no shared secret can be established because any bus eavesdropper can determine their value, by observing cluster contents. This case corresponds to the gray rows of Table II.
4) If $X = 0$, then it is known that $A = \overline{B}$, but their value cannot be determined by the eavesdropper since the two possible cases generate exactly the same bus traffic, as shown in the two middle rows of the table. Hence, the two nodes can establish a 1-bit secret $K = A = \overline{B}$.

It should be noted that, after a successful protocol run, R_A and R_B possess K and $\overline{K}$, respectively. Therefore, the protocol is not symmetric and it is important to tell their roles

apart. This is not an issue because the CAN XR protocol already supports nodes with multiple roles in a transaction. For instance, R_A can be defined to also take the role of I (and S), while R_B is a plain responder.

The extension to multi-bit secrets is very simple and consists of n protocol runs, each one attempting to establish a 1-bit secret as described above. For what concerns bus traffic related to the distributed XNOR operation, the n runs are performed as outlined in Section III, by sharing a single frame up to its capacity. If needed, it is then possible to use multiple frames according to suitable fragmentation and reassembly mechanisms like [11], [12].

B. Probability of Success

Under the assumption that bit values 0 and 1 are equiprobable, and A and B are statistically independent, all rows of Table II have the same probability of occurrence. Hence, a single protocol run succeeds in establishing a 1-bit shared secret with probability $p = \frac{1}{2}$.

We can define the protocol efficiency as the average ratio between the number of secret bits established in n runs, expressed by the random variable $Q(n)$, and the number of bits exchanged, namely $2n$. As also stated in [10], it is

$$E\left[\frac{Q(n)}{2n}\right] = \frac{E[Q(n)]}{2n} = \frac{pn}{2n} = \frac{1}{4}. \tag{3}$$

However, a more interesting formulation of the problem from the practical point of view is to determine what is the minimum number of protocol runs $n_0(k, \rho)$ of $n \geq k$ that ensures the establishment of $Q(n) \geq k$ secret bits with confidence probability ρ. In formula, we want to determine

$$n_0(k, \rho) = \min_n (P\left[Q(n) \geq k\right] \geq \rho), \;\; n \geq k. \tag{4}$$

This can easily be done by observing that $Q(n)$ is a binomial random variable corresponding to n Bernoulli experiments with probability of success $p = \frac{1}{2}$. Denoting with $F(k; n, \frac{1}{2})$ the well-known cumulative distribution function (CDF) of $Q(n)$, it is

$$\begin{aligned} P\left[Q(n) \geq k\right] &= 1 - P\left[Q(n) < k\right] = 1 - P\left[Q(n) \leq k-1\right] \\ &= 1 - F(k-1; n, \frac{1}{2}). \end{aligned} \tag{5}$$

Substituting (5) into (4) we eventually obtain

$$n_0(k, \rho) = \min_n (F(k-1; n, \frac{1}{2}) \leq 1 - \rho), \;\; n \geq k. \tag{6}$$

Table III lists the values of n_0 for several commonly used values of k and two different values of ρ. The same table also lists the overall number of frames needed for key establishment

$$M(n_0) = \left\lceil \frac{\lceil \frac{2n_0}{8} \rceil}{63} \right\rceil, \tag{7}$$

under the hypothesis that frames of length up to 64 bytes (the maximum frame size currently supported in CAN [1]) are used, in which one byte is reserved for control information related to fragmentation and reassembly. All frames are of maximal size, with the possible exception of the last one, whose length in bytes is

$$L(n_0) = \left\lceil \frac{2n_0}{8} \right\rceil - 63(M(n_0) - 1) + 1. \tag{8}$$

The values of $L(n_0)$ calculated in (8) neglect that not all frame lengths above 8 are allowed in CAN [1]. Hence, it may be necessary to use a larger length than strictly needed. The corrected value is shown between parentheses in Table III. As we can see, just 5 frame exchanges are needed to establish a 512-bit shared key with at least 99.9% of success. Hence, protocol overhead is considered to be acceptable.

TABLE III
VALUES OF n_0, $M(n_o)$, AND $L(n_0)$ FOR COMMON k AND ρ

	$\rho = 0.99$				$\rho = 0.999$			
k	n_0	M	$L\,[B]$		n_0	M	$L\,[B]$	
128	295	2	12	(12)	310	2	16	(16)
256	567	3	17	(20)	586	3	22	(24)
512	1101	5	25	(32)	1127	5	31	(32)

V. PROTOTYPE IMPLEMENTATION AND EVALUATION

The protocol proposed in Section III has been evaluated experimentally by means of a prototype implementation, in terms of correctness, memory footprint and execution time.

A. Experimental Setup

The setup used for the evaluation consists of 3 CAN nodes, namely, one initiator/responder I/R_A (which also acts as S), one responder R_B, and one consumer C. Both I/R_A and R_B are implemented by means of a *software-defined* CAN controller (SDCC), extended to support CAN XR, as explained in Section II and [9]. SDCC consists of several layered modules, whose structure closely reproduces the one specified by the CAN standard [1].

Unlike a regular CAN controller, which does not leave room for extension at/below the data-link layer, because it implements those layers in hardware, SDCC can be easily modified at will. In addition to the functions implemented in a normal CAN controller, it also includes extra support for role-dependent transmission/reception and takes particular care of bit synchronization and stuffing as they become much tricky when multiple nodes are involved in the same data field. The only hardware components needed by SDCC are a timer, a General-Purpose Input-Output (GPIO) port, and a transceiver. The timer drives SDCC real-time operations, while the GPIO port enables SDCC to communicate with the off-chip transceiver. In turn, the transceiver takes care of the physical-layer electrical interface to the CAN bus.

On the other hand, C plays a passive role in CAN XR transactions. It makes use of a standard, *hardware* CAN controller (HCC) to verify that bus traffic still conforms to the ISO 11898 standard [1], and hence, CAN XR is backward

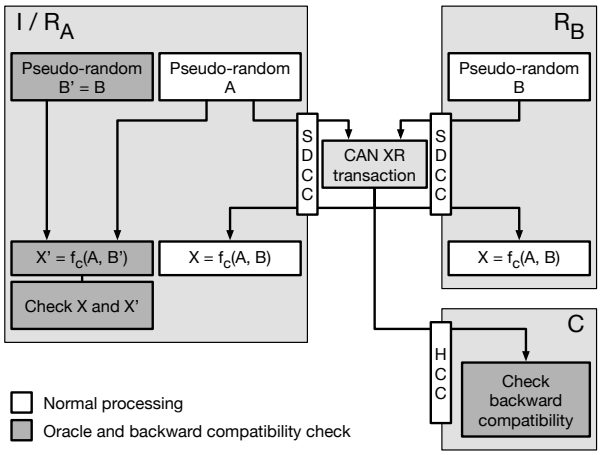

Fig. 3. Experimental setup.

TABLE IV
MEMORY FOOTPRINT BY MODULE (BYTES)

Module	*Text and RO data*	*RW init. data*	*BSS data*
I/R_A MAC extension and PCBs	877	0	264
R_B MAC extension and PCBs	460	0	244
General protocol implementation	*1316*	*0*	*0*
CAN medium access control (MAC)	1728	0	0
Physical coding sub-layer (PCS)	376	0	0
Platform-independent PMA	16	0	0
Timer and GPIO PMA	324	0	0
Total (excl. runtime library modules)			
I/R_A	4637	0	264
R_B	4220	0	244
Total (incl. runtime library modules)			
I/R_A	44692	1292	10000
R_B	44180	1292	9980

compatible. All nodes are based on a LPC1768 microcontroller [14], whose Cortex-M3 CPU runs at 100 MHz. In order to cope with current SDCC processing time limitations on the aforementioned CPU, the CAN bit rate has been set to 50 kb/s, one of the standard rates specified by the CANopen standard [4].

As shown in Fig. 3 (white blocks), I/R_A and R_B implement the protocol for calculating arbitrary Boolean functions proposed in Section III. More specifically, they start from two pseudo-random values, namely A and B. Each bit of them is extended to a 4-bit sequence as indicated in Fig. 2 and then concatenated to serve as the data field of a CAN XR frame to be exchanged on the CAN bus. As aforementioned, SDCC is in charge of frame formatting as well as simultaneous transmission and synchronization when more than one node takes part in sending the data field.

Besides, I/R_A checks that protocol results (achieved via the white blocks) are correct by means of an *oracle* (gray blocks). The oracle is able to predict the pseudo-random material that R_B shall generate at each protocol round and use it to calculate the expected outcome X'. It then compares X' with the actual outcome X established by the protocol.

The same setup can also be used to realize the key establishment protocol proposed in [10], by specializing the structure shown in Fig. 2 according to the reformulation discussed in Section IV. However, this possibility has not been further explored in this paper for conciseness and is planned as an upcoming work.

B. Results

The experimental evaluation aimed, first of all, at confirming the correctness of the proposed protocol and its implementation, beyond the theoretical reasoning presented in Section III. To this purpose the experimental setup outlined in Fig. 3 was configured to perform a total of 768000 protocol rounds in several hours of runtime. Upon each round I/R_A and R_B randomly select one point within the operand space, consisting of 2^{36} elements (two 16-bit operands A and B, plus the 4-bit function selector c).

Since the random number generator produces uniformly distributed values, the operand space is explored according to the classic Monte Carlo method [15]. The number of collected samples corresponds to $1.12 \cdot 10^{-5}$ of the space, which is deemed to be sufficient to double-check correctness. Further experiments involving an even larger number of samples are in progress and no failures were detected so far.

Afterwards, the firmware was statically inspected to assess its memory footprint, yielding the figures listed in Table IV. Footprint has been broken down by module and divided into three categories: code (traditionally called *text*) and read-only data, read-write initialized data, and read-write uninitialized data (traditionally called *BSS data*). This is because in an embedded system they may correspond to different kinds of memory (typically, Flash memory versus static RAM).

As shown in the third row of the table, the general protocol implementation accounts for about 28% and 31% of the total I/R_A and R_B text sizes, respectively, while it does not contain either read-write or BSS data. At the same time, the total firmware text sizes compare very favorably to the amount of Flash memory available on the microcontroller in use (less than 5 KB used on each node versus 512 KB available, that is, about 1%). The total data sizes are also negligible with respect to the available static RAM (less than 300 B used versus 64 KB available, about 0.5%). It should also be noted that the size of the I/R_A and R_B MAC extension modules, given in the first two rows of Table IV, includes all protocol control blocks (PCBs) needed by SDCC as BSS data.

Those results confirm that the proposed approach can easily be integrated with existing, low-cost firmware designs without concerns about overflowing memory capacity. The overall firmware image sizes (last two rows of Table IV) are significantly larger because they include C library modules linked in by the `printf` function, used to log test results and debugging information. However, these modules will likely not be included in production firmware.

Last, but not least, the firmware was instrumented to evalu-

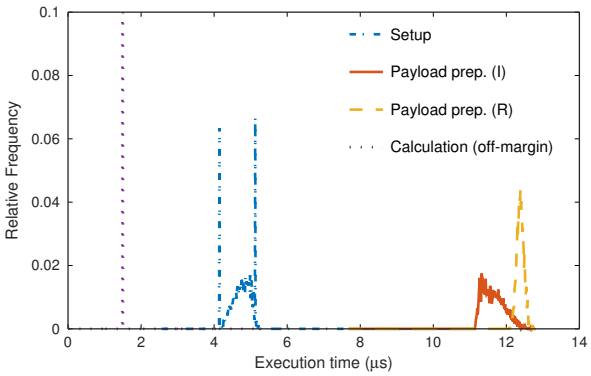

Fig. 4. Execution time of main protocol functions.

ate the execution time of the general protocol implementation, by means of a 32-bit timer running at 100 MHz. The execution time of the main protocol functions was evaluated by calculating the differences between timer readings taken immediately after/before calling the function and collecting them into a histogram for 10000 protocol rounds. Code inspection shows that data collection requires about 10 assembly language instructions, leading to an estimated overhead under $0.5\,\mu$s. The bus transmission time has not been evaluated because it depends on the CAN bit rate rather than software performance.

Results are depicted in Fig. 4. The majority of the time is spent in payload preparation (about $12\,\mu$s) and setting up data that are needed by other protocol functions and depend on c (about $4.5\,\mu$s). It is however worth noting that data setup overheads can be reduced when running multiple protocol rounds with the same c, because the setup must be carried out only once. Moreover, overheads can be avoided altogether if c is constant and known in advance, since the setup can be performed at compile time in this case.

The calculation of f_c after the frame exchange is the fastest operation and it takes less than $2\,\mu$s. Overall, this leads to a total protocol execution time of $12 + 4.5 + 2 = 18.5\,\mu$s, which is a small fraction of the bus transmission time even considering the highest bit rate supported by classic CAN, that is, 1 Mb/s. All protocol functions, except f_c calculation, are affected by an amount of data-dependent jitter between 1 and $2\,\mu$s. No attempt to reduce the jitter has been made because it was deemed irrelevant for this kind of application.

VI. Conclusion

This paper illustrates how CAN XR—a backward-compatible extension of the CAN data-link layer—can be leveraged to calculate arbitrary binary Boolean functions, in a distributed way and without necessarily disclosing their operands. As a special case, it was shown that CAN XR can effectively implement a key establishment protocol formerly appeared in literature [10], which also paves the way to support other security protocols.

The experimental evaluation of a prototype implementation (SDCC) further substantiates that the proposed method works correctly and meets the typical memory footprint and execution time requirements of low-cost embedded systems. The SDCC implementation has been carried out in analogy with the well-established concept of software-defined radio [16]. With respect to other software simulators proposed in literature [17], SDCC's advantage is that it operates completely in real time. Hence, it can communicate with hardware-based controllers directly for cogent correctness and compatibility checks.

As future work, it is foreseen to extend the method to support n-ary Boolean functions, assess its applicability to other scenarios, and better investigate its operand non-disclosure properties, also with the help of further improvements to SDCC.

Acknowledgment

This research has been supported in part by the US National Science Foundation (CNS Grant No 1646317).

References

[1] ISO, *ISO 11898-1:2015 – Road vehicles – Controller area network (CAN) – Part 1: Data link layer and physical signalling*, International Organization for Standardization, Dec. 2015.
[2] SAE, *SAE J1939/21 – Data Link Layer*, SAE International, Dec. 2010.
[3] ISO, *ISO 15031-5:2015 – Road vehicles – Communication between vehicle and external equipment for emissions-related diagnostics – Part 5: Emissions-related diagnostic services*, International Organization for Standardization, Aug. 2015.
[4] CiA, *CiA 301 V4.2.0 – CANopen application layer and communication profile*, CAN in Automation e. V., Feb. 2011.
[5] K. Koscher, A. Czeskis, F. Roesner, S. Patel, T. Kohno, S. Checkoway, D. McCoy, B. Kantor, D. Anderson, H. Shacham, and S. Savage, "Experimental security analysis of a modern automobile," in *Proc. IEEE Symposium on Security and Privacy (SP)*, 2010, pp. 447–462.
[6] Y. Burakova, B. Hass, L. Millar, and A. Weimerskirch, "Truck hacking: An experimental analysis of the SAE J1939 standard," in *Proc. 10th USENIX Workshop on Offensive Technologies (WOOT)*, 2016, pp. 1–10.
[7] P. Marino, F. Poza, M. A. Dominguez, and S. Otero, "Electronics in automotive engineering: A top–down approach for implementing industrial fieldbus technologies in city buses and coaches," *IEEE Transactions on Industrial Electronics*, vol. 56, no. 2, pp. 589–600, Feb. 2009.
[8] 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," in *Proc. 20th USENIX Conference on Security (SEC)*, 2011, pp. 1–16.
[9] G. Cena, I. Cibrario Bertolotti, T. Hu, and A. Valenzano, "CAN XR: CAN with eXtensible in-frame Reply," in *Proc. of 14th IEEE Intl. Conference on Industrial Informatics (INDIN)*, Jul. 2016, pp. 1198–1201.
[10] A. Mueller and T. Lothspeich, "Plug-and-secure communication for CAN," in *Proc. of the Intl. CAN Conference (iCC)*, Oct. 2015, pp. 06-6–06-14.
[11] G. Cena, I. Cibrario Bertolotti, T. Hu, and A. Valenzano, "Design, verification, and performance of a MODBUS-CAN adaptation layer," in *Proc. 10th IEEE International Workshop on Factory Communication Systems (WFCS)*, May 2014, pp. 1–10.
[12] ——, "Seamless integration of CAN in intranets," *Computer Standards & Interfaces*, vol. 46, pp. 1–14, May 2016.
[13] G. Cena and A. Valenzano, "On the properties of the flexible time division multiple access technique," *IEEE Transactions on Industrial Informatics*, vol. 2, no. 2, pp. 86–94, May 2006.
[14] *LPC17XX User manual, UM10360 rev. 2*, NXP B.V., Aug. 2010.
[15] D. E. Knuth, *The Art of Computer Programming, Volume 2: Seminumerical Algorithms*, 3rd ed. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1997.
[16] T. Ulversoy, "Software defined radio: Challenges and opportunities," *IEEE Communications Surveys Tutorials*, vol. 12, no. 4, pp. 531–550, fourth quarter 2010.
[17] P. Mundhenk, A. Mrowca, S. Steinhorst, M. Lukasiewycz, S. A. Fahmy, and S. Chakraborty, "Open source model and simulator for real-time performance analysis of automotive network security," *SIGBED Rev.*, vol. 13, no. 3, pp. 8–13, Aug. 2016.