LiBrA-CAN: Lightweight Broadcast Authentication for Controller Area Networks

BOGDAN GROZA and STEFAN MURVAY, Politehnica Timisoara
ANTHONY VAN HERREWEGE and INGRID VERBAUWHEDE, KU Leuven

Despite realistic concerns, security is still absent from vehicular buses such as the widely used Controller Area Network (CAN). We design an efficient protocol based on efficient symmetric primitives, taking advantage of two innovative procedures: splitting keys between nodes and mixing authentication tags. This results in a higher security level when compromised nodes are in the minority, a realistic assumption for automotive networks. Experiments are performed on state-of-the-art Infineon TriCore controllers, contrasted with low-end Freescale S12X cores, while simulations are provided for the recently released CAN-FD standard. To gain compatibility with existent networks, we also discuss a solution based on CAN+.

Categories and Subject Descriptors: C.2.2 [**Computer-Communication Networks**]: General: Security and protection

General Terms: Security, Algorithms, Performance

Additional Key Words and Phrases: CAN bus, cryptography, authentication, broadcast

ACM Reference Format:
Bogdan Groza, Stefan Murvay, Anthony Van Herrewege, and Ingrid Verbauwhede. 2017. LiBrA-CAN: Lightweight broadcast authentication for controller area networks. ACM Trans. Embed. Comput. Syst. 16, 3, Article 90 (April 2017), 28 pages.
DOI: http://dx.doi.org/10.1145/3056506

1. MOTIVATION AND RELATED WORK

Vehicular network security has established itself as an intense research topic in the last few years. Outstanding experimental results from Koscher et al. [2010] and later Checkoway et al. [2011] showed vehicles to be easy targets for malicious adversaries. The myriad of attacks reported in the last 5 years showed that virtually any subsystem inside a car is vulnerable to attacks that exploit the Controller Area Network (CAN) bus as an entry point. To justify our concern, Table I summarizes some of the attacks reported so far, grouping them according to the vehicular subsystem on which they act: power train, chassis, body, multimedia, telematics, HMI, or active/passive safety. These attacks are reproducible on most (if not all) cars on the market. Any

This work was supported in part by research grant POSDRU 107/1.5/S/77265, inside POSDRU Romania 2007-2013. It was also supported in part by the Research Council KU Leuven: GOA TENSE (GOA/11/007). In addition, this work was supported in part by the Flemish Government through FWO G.0550.12N and the Hercules Foundation AKUL/11/19, and by the European Commission through the ICT program under contract FP7-ICT-2011-284833 PUFFIN.
Authors' addresses: B. Groza and S. Murvay, Faculty of Automatics and Computers, Politehnica University of Timisoara, Bd. V. Parvan nr. 2, Timisoara, Romania; emails: bogdan.groza@aut.upt.ro, stefan.murvay@gmail.com; A. van Herrewege and I. Verbauwhede, ESAT/COSIC and IMEC, KU Leuven, Kasteelpark Arenberg 10, bus 2452, 3001 Heverlee, Belgium; emails: anthony@anthonyvh.com, ingrid.verbauwhede@esat.kuleuven.be.

DOI: http://dx.doi.org/10.1145/3056506

Table I. Recently Reported Attacks Having CAN as Entry Point

Subsystem	Affected module	Consequence
Power Train (longitudinal propulsion: engine, transmission, etc.)	Engine	Increase idle RPM, temporary RPM increase, initiate crankshaft re-learn (disturbs engine timings), disable cylinders, kill engine, grind starter, remote car start, cannot turn on (DoS to/from BCM), cannot turn off (while turned on, cause BCM to activate ignition output) [Koscher et al. 2010]
Chassis (wheels and their relative position and movement: steering, braking, etc.)	Brakes	Disable, engage front left, engage front right, unlock front left, unevenly engage right brakes, releases brakes (prevents braking) [Koscher et al. 2010]
	Power steering	Disable [Koscher et al. 2010]
Body (entities that do not belong to vehicle dynamics: wipers, lighting, window lifter,	Doors	Lock/unlock car, unlock all while at speed [Koscher et al. 2010]
	Gauges & instruments	Falsify speedometer reading, falsify fuel level, freeze display panel [Koscher et al. 2010]
	Electric window lift	Control windows [Hoppe and Dittman 2007], DoS [Hoppe et al. 2008], disable window relays [Koscher et al. 2010]
Multimedia, Telematics and HMI (information exchange: display, switches, radio, navigation, Internet, etc.)	Radio	Increase volume, change display, produce ticking sound [Koscher et al. 2010]
	Driver Information Center	Change display [Koscher et al. 2010]
Active/Passive Safety (airbags, warnings, seatbelt, ABS, ESP, cruise control, etc.)	Airbag	Suppress missing airbag warnings by emulating its presence [Hoppe et al. 2008]

device inside a car can be seriously affected by real-world adversaries, and this is no surprise as long as CAN (the bus used to connect all relevant communications inside a vehicle) lacks cryptographic security. The same holds for other communication buses inside the car, even for the most recent developments (e.g., FlexRay, CAN-FD).

Proposals for ensuring security for in-vehicle buses are not many, and there are several clear reasons behind this. First, the relevance of ensuring security inside vehicles was decisively proved only in the recent years [Koscher et al. 2010; Checkoway et al. 2011]. Second, some of the design principles used by manufacturers are out of reach for the academic community, making it hard to understand many assertions behind protocol designs. Since intravehicle communication systems are subject to many constraints that range from the technical side up to the economical factors (e.g., hard real-time constraints, low-cost margins, etc.), exploring for new solutions is vital. Several approaches to in-vehicle security advocate the use of secure gateways between different ECUs (Electronic Control Units) or subnetworks [Bar-El. 2009; Wolf et al. 2006] and rely on basic cryptographic constructions (encryptions, signatures, etc.). These proposals do not particularly target broadcast authentication on CAN, which is still the most common communication bus inside vehicles; we discuss protocols dedicated for CAN bus authentication in the following subsection.

1.1. Related Work

TESLA and CANAuth. TESLA-like protocols proved to be highly effective in sensor networks [Perrig et al. 2000, 2001] and so far are the most efficient alternative for ensuring broadcast authentication with low-cost symmetric primitives (e.g., Message Authentication Codes (MACs)). However, when it comes to the CAN bus, this protocol family has one drawback that is critical for automotive systems: delays, which by the nature of TESLA are unavoidable. The main purpose of the work in Groza and Murvay [2013] is to determine a lower bound on these delays and establish some tradeoffs. Delays in the order of milliseconds, as shown to be achievable in Groza and Murvay [2013], are satisfactory for many scenarios, but such delays do not appear to be small enough for in-vehicle communication. Due to the nature of TESLA-like protocols, these delays are always present and cannot be eliminated. To reduce the delays, one can use a bus with a higher throughput, more computational power, and better electronic components (e.g., oscillators), but this will greatly increase the cost of components, nullifying in this way the cost effectiveness of CAN. CANAuth [Van Herrewege et al. 2011] is a protocol that has the merit to follow in great detail the specifications of CAN; its security is specifically designed to meet the requirements of the CAN bus. In particular, CANAuth is not intended to achieve source authentication as the authentication is bound to the message IDs and messages may originate from different sources, which will be impossible to trace. This fits the specification of CAN, which has a message-oriented communication. However, a first issue is that the number of CAN IDs is quite high, in the order of hundreds (11 bits) or even millions in the case of extended frames (29 bits), and storing a key for each possible ID does not seem to be so practical. For this purpose, in Van Herrewege et al. [2011], a clever solution is imagined: the keys are linked to multiple ID codes using masks, which greatly reduces the number of keys. But still, this leads to some security concerns. Traditionally, keys are associated to entities to ensure that they are not impersonated by adversaries, but by associating keys to messages, the identity of the sender can no longer be verified. For example, any external tool (assume On-Board Diagnostics (OBD) tools, which are widespread) that is produced by external third parties will have to embed the keys associated for each ID that it sends over or even just listens to on CAN. Obviously, if such a device is easy to compromise (even an innocuous one such as a passive receiver), then all the IDs it was allowed to authenticate are equally compromised.

Voting. Szilagyi and Koopman [2009, 2010] introduce a validity voting scheme. The scheme is intended for generic time-triggered communication such as TT-CAN, FlexRay, and so forth. The core part of the protocol relies on the classical paradigm of sharing keys between each sender and receiver, then authenticating packets on one MAC per receiver basis. Further, to make it feasible to embed the MACs in a single frame, the tags are truncated and concatenated (e.g., three MACs each of 8 bits are fitted at the end of a single frame). The communication is time triggered, each receiver releasing his or her message and vote on previous messages in fixed time slots. Both the new message and the receiver's vote, along with all previously received messages, are authenticated under the same array of MACs to other receivers. The procedure leads to a drawback as stated in Szilagyi and Koopman [2010]: for frames that are lost, the receive history of the nodes does not match and authentication will fail for these frames. As suggested in Szilagyi and Koopman [2010], this can be fixed by adding additional bits for lost packets, but sufficient votes from other nodes would still be required to deem the frame authentic.

Other proposals. Since our initial conference report on LiBrA [Groza et al. 2012], several other lines of work have appeared. CaCAN [Kurachi et al. 2014] uses a centralized approach in which a central node monitors the bus for malicious frames and destroys them with error flags. As acknowledged by the authors, one drawback of this approach is that the monitor node can be unplugged from the bus, nullifying the protection layer. MaCAN [Hartkopp et al. 2012] uses standard key-sharing between nodes and an AES-based CMAC for authentication; the authentication tag is truncated to 32 bits in order to fit it along with the message in a single frame. Subsequent formal analysis [Bruni et al. 2014] found the protocol vulnerable to some attacks but easy to fix. One more serious security issue with MaCAN is that the use of a counter is avoided and freshness is ensured by linking to a common time value preserved on the nodes. Needless to say, synchronization errors will always be present and the protocol appears to provide no protection against adversaries that can force such errors by delaying the synchronization packets from the time server. This means that replay attacks cannot be completely avoided in MaCAN. Moreover, messages are authenticated only for a fixed

Table II. Specifications, Advantages, and Limitations of Current Proposals for Ensuring CAN Security

Protocol	Specifications and advantages	Possible Limitations
TESLA-CAN [Groza and Murvay 2013]	- efficient use of symmetric primitives with time synchronization - successful and well studied in sensor networks	- fixed authentication delays (usually milliseconds), - time triggered release of keys (potential conflicts with CAN arbitration) - resynchronization can be an issue at very small authentication delays
CAN-Auth [Van Herrewege et al. 2011]	- ID oriented authentication, follows in detail CAN specifications - no authentication delays or time synchronization	- no source authentication (unclear security implications, e.g., third party tools can inject forged frames) - number of keys increases with number of IDs
Voting [Szilagyi and Koopman 2010]	- shared symmetric keys between each 2 nodes - each node votes for the authenticity of previous messages	- small number of nodes - time triggered release of authentication tags (nodes need to wait over multiple time slots to get sufficient votes) - nodes are required to be present and vote - disagreements between nodes if packets are lost - each node needs to apply a MAC on his current message, his votes on previous messages, and all messages that he received previously
LiBrA-CAN (this work)	- efficient source authentication with dishonest minority - efficient forgery detection with MAC mixing - no authentication delays or time synchronization	- small number of participants - malicious nodes in minority

number of times; this leaves the possibility for an adversary to compromise frames that are missing the authentication tag. The use of shared keys between nodes is also explored in Woo et al. [2015], Wang and Sawhney [2014], and Bittl [2014]. Other works such as Lin et al. [2015] are focused on the impact of the added security mechanisms on real-time constraints. A survey on security threats and protocols is available in Studnia et al. [2013].

1.2. LiBrA-CAN, Design Intentions

Scope. This work extends and improves our conference report from Groza et al. [2012]. We include more motivation, details, and proofs that were previously omitted due to the obvious space constraints. The main LiBrA scheme is changed from the centralized version to a distributed scheme that is more appropriate to the context of CAN, and new variations of the scheme are further provided. We also bring improvements and details on the computational costs of the mixed MAC construction, which gives security improvements to LiBrA. Finally, we provide in this extended version a more detailed performance analysis, taking into account the recently released CAN-FD [Robert BOSCH GmbH 2012] standard (not available at the time of our initial report).

LiBrA-CAN is based on two paradigms: key splitting and MAC mixing. The latter procedure is optional and is intended to increase security by allowing each node to detect a potential forgery. Key splitting allows a higher entropy for each mixed MAC that is sent at the cost of losing some security for groups that contain more malicious nodes. An adversarial majority will be required to break the protocol, while if there are fewer adversarial nodes, the security level is drastically increased. Consequently, this appears to give a flexible and efficient tradeoff. Note that in contrast to the scheme of Szilagyi and Koopman, which requires the nodes to be present and vote, LiBrA benefits from a majority of noncorrupted nodes but does not require their presence to vote (it is just their keys that need to be safe). This procedure is not new; similar techniques were proposed in the past in the context of broadcast security. We could trace this back to the work of Fiat and Naor [1994], but there are many papers on this subject. The work of Canetti et al. [1999] provides efficient constructions based on the same principles. However, the constraints of our application in CAN networks are entirely different from related work where this procedure was suggested or used in scenarios such as sensor networks [Chan et al. 2003], pay-TV [Naor and Pinkas 1998], and so forth. The main

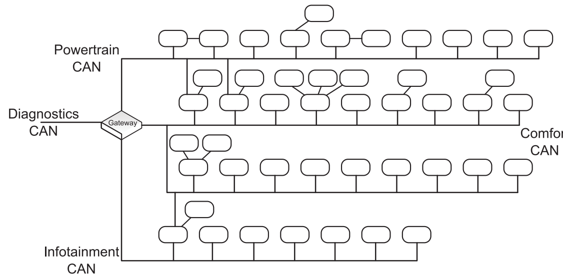


Fig. 1. Network topology of a VW Phaeton based on Leohold [2004].

idea behind such schemes is that groups of l corrupted receivers cannot learn the secret (in settings with $n > l$ users). In addition to this, we exhibit a distinct contribution in the construction of Linearly Mixed MACs, which allows us to amalgamate several authentication tags in a single tag via a system of linear equations. This construction has the advantage that if one of the MACs is wrong, then this will affect all other MACs and thus the mixed MAC will fail to verify on any of the multiple keys. This increases the chance of a forgery being detected, and ultimately it increases the reliability in case benign nodes are in possession of a wrong key. To the best of our knowledge, this procedure is new. The closest work that we could find consists of the multiverifier signatures proposed by Roeder et al. [2012]. In their work, linear systems of equations are used as well upon message authentication codes, but the security properties and goals of their construction are different. For our construction, we require that the mixed MACs are strongly nonmalleable, a property that appears to be entirely different.

Another assumption behind our design is that the number of nodes that are connected to the same bus line is not large. To strengthen our assumption, we present in Figure 1 the network topology of a high-end vehicle based on Leohold [2004]. Indeed, it is easy to see that while there is a high number of ECUs, not all of them share the same network, and consequently they can be easily placed into small broadcast groups based on the subnetwork they are part of. While indeed ECUs inside cars come from different manufacturers that may or may not be trustworthy, we believe that suspicious ECUs should be limited in number, since the potential insertion of a trapdoor in some component will discredit the public image of the manufacturer too much and there appears to be little or no benefit for this. Moreover, ECUs that come from the same manufacturer and are trustworthy with each other may potentially use the same shared key (randomly generated at runtime for each subnetwork in which they are plugged). In this way, the number of actual keys needed to ensure broadcast security should be smaller than it appears to be at first sight. In our design, we try to take advantage of this assumption, and our approach is especially efficient in the case when compromised nodes form only a minority. The idea of malicious nodes in minority is also supported by research subsequent to our initial conference report on LiBrA-CAN [Groza et al. 2012]. Most of the research works suggest that the attack comes either from a malicious device inserted on the bus or from compromising the software inside a single ECU [Kurachi et al. 2014; Woo et al. 2015; Wang and Sawhney 2014]. Grouping nodes under the same secret shared key is also proposed in Hartkopp et al. [2012], where groups are formed on the trust level on the node.

It is also an advantage for the current proposal that a new CAN standard that supports flexible data rates was released: CAN-FD [Robert BOSCH GmbH 2012]. The larger data frames of CAN-FD [Robert BOSCH GmbH 2012] allow us to design a more robust authentication protocol that significantly reduces the overhead of independent authentication frames; we detail this in the main version of the protocol. Besides the

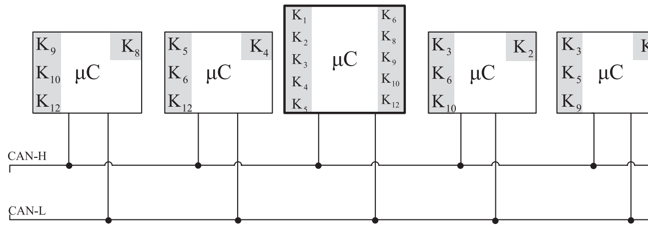


Fig. 2. Topology of the CAN bus and key allocation for centralized authentication.

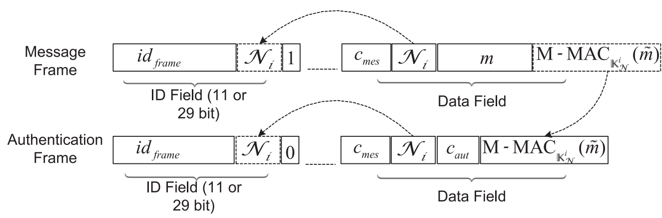


Fig. 3. Data frames and authentication frames.

larger data field, CAN-FD allows an increased data rate after the arbitration, which will make it even faster to carry the authentication data. We present real-time simulation of our protocol using the industry-standard CANoe tool from Vector (www.vector.com).

2. THE PROTOCOL

We begin with a brief overview of the application setting and assumptions. Then we outline the main authentication scheme and discuss some variations or improvements to it.

2.1. Network Topology and Frame Structure

The CAN is a broadcast serial bus designed by Bosch [Robert BOSCH GmbH 1991; International Organization for Standardization 2003]. The typical topology consists of a differential bus that connects multiple nodes by two wires (called CAN-High and CAN-Low). This network architecture is suggested in Figure 2; the setup corresponds to one of the protocol versions that we discuss in Section 2.5 for which we defer the details on key allocation.

CAN frames carry at most 8 bytes of data. Each CAN frame begins with a start bit followed by the arbitration field (29 bits in extended frames and 11 bits in standard frames), a control field (6 bits), data bits (0 to 64), CRC sequence (15 bits), a 2-bit acknowledgment, and 7 bits that mark the end of the frame. Stuffing bits are added after each 5 consecutive bits of identical value. The newer CAN-FD standard [Robert BOSCH GmbH 2012] allows up to 64 bytes of data to be carried by one frame and more, the data rate can be increased after the arbitration procedure.

For the case of standard CAN frames (unable to carry both the data and authentication tag), as well as for some variations of the main scheme, we separate between message frames and authentication frames. Larger data blocks or authentication tags (exceeding 8 bytes) can be split across multiple frames with the same ID field and counter. On the other side, with CAN-FD frames, it is advantageous to embed the authentication tag in the message frame and take benefit of the increased data rate that follows the arbitration procedure. This also reduces the authentication delay and allows immediate verification.

In Figure 3, we suggest the structure of the frame for the case when the authentication tag, that is, $\mathsf{M\text{-}MAC}_{\mathbb{K}^i_{\mathcal{N}}}(\widetilde{m})$, is embedded in the message frame, and we also outline the case when it is sent as a separate authentication frame (dashed arrow). In both cases, the frame structure consists of the identifier of the message id_{frame}, which is the usual CAN ID; the identifier of the source node $\mathcal{N}_i$; a message counter c_{mes}; the message itself m; and the authentication tag $\mathsf{M\text{-}MAC}_{\mathbb{K}^i_{\mathcal{N}}}(\widetilde{m})$. Supplementarily, in the case of authentication frames, a new counter c_{aut} specifies the number of the authentication frame (intended for protocol variants where there is more than one authentication frame for a message). The last bit of the identifier field specifies whether a frame carries an authentication tag or message (1 vs. 0). Separate authentication frames are sent in our experimental setup with CAN-capable boards, while in the CAN-FD simulation from CANoe the data frames carry the authentication tag as well (allowing immediate authentication). Further adjustments can be done. For example, since the data field is quite short, the node identifier (which denotes the source of the authentication frame) can be moved in the ID field (suggested by a dashed arrow in Figure 3). A discussion dedicated to the use of counters for ensuring freshness follows in the next subsection.

2.2. Adversary Model and Security Objectives

In the previously described setting, we assume the usual presence of an adversary that has full control over the communication channel. That is, the adversary can eavesdrop, modify, block, or send messages at will.

The security objective of our scheme is to ensure an authentic channel, that is, to prevent messages that originate from the adversary to be accepted by the honest principals. This is achieved by adding to each message a number of authentication tags that are computed with secret keys that are shared between nodes. By increasing the size of the authentication tags, the scheme can essentially resist any number of corrupted nodes, as long as a single key remains secure on the sender's and receiver's side. For practical reasons, it is our assumption that corrupted nodes are in the minority and thus we hold as an intuitive model that the adversary is a device that tries to act on behalf of an honest node from the network. In real-world instantiations, this may happen in one of the following two circumstances. The adversary can be an external device that is plugged into the network (this is the general cause of all the attacks reported so far, e.g., Koscher et al. [2010]), or it is one of the genuine ECUs from the network that tries to impersonate one of the other honest ECUs (this seems a plausible scenario in case of an untrusted third-party component). In the latter case, the adversary is in possession of a limited number of keys but still misses some of the keys that are held by the honest nodes (otherwise, if the adversary has all the keys, there is nothing that can be done). The design intention behind LiBrA-CAN is to make the scheme resilient in front of both these situations. The case of a larger number of adversaries (i.e., more nodes that leaked their keys) requires a quantitative discussion based on the main LiBrA-CAN protocol and is deferred for Examples 2 and 3 in Section 2.4 (however, none of the practical attacks reported so far relied on such an assumption).

As a particular flavor of authentication, ensuring the freshness of each message is a desired goal, that is, preventing replay attacks. We do believe that this is a critical issue for real-time communication networks that requires dedicated research; here we provide only a generic security construction. Related work is split around the use of counters [Kurachi et al. 2014; Woo et al. 2015; Wang and Sawhney 2014] and the use of timestamps [Hartkopp et al. 2012]. The use of timestamps cannot fully prevent replays since synchronization errors cannot be completely removed. There is no concrete specification in Hartkopp et al. [2012] for the periodicity of the timestamps; the consequences of a replay attack within the timeframe given by the synchronization error are unclear. Moreover, we note that the synchronization protocol from Hartkopp et al. [2012] is in fact insecure since an adversary can delay the response of the time server, making the ECU receive a somewhat older reference time (the protocol does not check for synchronization errors at all and small errors will always be present). Counters remove part of this problem (replays during the timeframe given by the synchronization error), but the drawback in using counters is that nodes can go into bus-off mode or disconnect from the bus; in this way, the latest value of the counter is lost. Still, having in mind that synchronization requires a protocol that is periodically triggered, checking for synchronization errors, and that replays may still be possible, we prefer the use of counters. We assume that whenever the node rejoins the bus, for example, after restart, the correct value of the counter will be obtained. Updating the counter can happen similarly to the way synchronization is done but hopefully it will be done sporadically (improving on the use of bandwidth) and will always prevent a message from being accepted twice, that is, replay. If the delayed acceptance of a message can be a security problem, then adding timestamps as well is unavoidable, as counters ensure uniqueness for a received message but an adversary can always delay the message up until the counter is updated by some other receive event. The size of the message counter c_{mes} strictly depends on the number of messages that are expected to be authenticated under the same key. For example, in the case of high-speed 1Mbps CAN, at most 10,000 to 20,000 messages are expected each second, and a 32-bit counter is more than sufficient for several days. The group keys can be refreshed on an hourly basis and the counters reset when this happens, thus significantly reducing the size of the counters. Moreover, this can be further improved by associating the counter to each pair formed by node (sender) identity and key, since we expect that messages from the same sender (i.e., reports for the same signals) are not sent quicker than 1 to 10ms intervals and the counter size could drop to 16 bits. This will save bandwidth, which should be more important than the additional memory required for storing more counters since these are small (e.g., 16 bits). In what follows, we assume that each node keeps an individual message counter for each sender-key pair. The counter is reset at each protocol restart and the counter of each received message is checked for monotonicity whenever a message is received from a particular sender. Following a successful receive event (a frame that passes the authentication test), the counter is updated.

2.3. Mixed Message Authentication Codes

In the main scheme, we make use of Mixed Message Authentication Codes (M-MACs), which amalgamate more MACs into one. The M-MAC uses an array of keys to build a tag that is verifiable by any of the keys. The easiest way to build an M-MAC is simply by concatenating multiple tags; such a construction is fine for our protocol and can be safely embodied in the main scheme (however, we do further improve on this trivial construction). The generic M-MAC construction is described next.

***Construction* 1** *(Mixed Message Authentication Code). A mixed message authentication code* M-MAC *is a tuple* (Gen, Tag, Ver) *of probabilistic polynomial-time algorithms such that:*

(*1*) $\mathbb{K} \leftarrow \mathsf{Gen}(1^\ell, s)$ *is the key generation algorithm that takes as input the security parameter* ℓ *and set size* s *and then outputs a key set* $\mathbb{K} = \{\Bbbk_1, \ldots, \Bbbk_s\}$ *of* s *keys.*
(*2*) $\tau \leftarrow \mathsf{Tag}(\mathbb{K}, \mathbb{M})$ *is the* MAC *generation algorithm that takes as input the key set* $\mathbb{K}$ *and message tuple* $\mathbb{M} = (m_1, \ldots, m_s)$*, where each* $m_i \in \{0,1\}^*$ *then outputs a tag* τ *(whenever needed, to avoid ambiguities on the message and key, we use the notation* $\mathsf{M\text{-}MAC}_{\mathbb{K}}(\mathbb{M})$ *to depict this tag).*
(*3*) $v \leftarrow \mathsf{Ver}(\Bbbk_i, m_i, \tau)$ *is the verification algorithm that takes as input a key* $\Bbbk_i \in \mathbb{K}$*, a message* $m_i \in \{0,1\}^*$*, and a tag* τ *and outputs a bit* v *that is* 1 *if and only if the tag is valid with respect to the key* $\Bbbk_i$ *and message* m_i*; otherwise, the bit* v *is* 0*. For correctness, we require that if* $\Bbbk_i \in \mathbb{K}$ *and* $m_i \in \mathbb{M}$*, then* $1 \leftarrow \mathsf{Ver}(\Bbbk_i, m_i, \mathsf{M\text{-}MAC}_{\mathbb{K}}(\mathbb{M}))$.

Properties. The first security property that we require for an M-MAC is *unforgeability* and it is a standard property for any MAC code (this property forbids the adversary from forging a correct authentication tag). We do develop on this by requiring a new property that we call *strong nonmalleability* and that lets any verifier detect whenever the adversary had tampered with any part of the M-MAC. We show that both these properties are achievable by the following LM-MAC construction, while simple concatenation of the tags fails in ensuring the second property.

***Construction* 2** (*Linearly Mixed MAC*). *We define the* LM-MAC *as the tuple of probabilistic polynomial-time algorithms* (Gen, Tag, Ver) *that work as follow:*

(*1*) $\mathbb{K} \leftarrow \mathsf{Gen}(1^\ell, s)$ *is the key generation algorithm that flips coins and returns a key set* $\mathbb{K} = \{\mathbb{k}_1, \ldots, \mathbb{k}_s\}$, *where each key has* ℓ *bits* (*ℓ is the security parameter of the scheme*).

(*2*) $\tau \leftarrow \mathsf{Tag}(\mathbb{K}, \mathbb{M})$ *is the* MAC *generation algorithm that returns a tag* $\tau = \{x_1, x_2, \ldots, x_s\}$, *where each* x_i *is the solution of the following linear system in* $GF(2^b)$:

$$\begin{cases} \mathsf{KD}_1(\mathbb{k}_1, m_1) \cdot x_1 + \cdots + \mathsf{KD}_s(\mathbb{k}_1, m_1) \cdot x_s \equiv \mathsf{MAC}_{\mathbb{k}_1}(m_1) \\ \mathsf{KD}_1(\mathbb{k}_2, m_2) \cdot x_1 + \cdots + \mathsf{KD}_s(\mathbb{k}_2, m_2) \cdot x_s \equiv \mathsf{MAC}_{\mathbb{k}_2}(m_2) \\ \cdots \\ \mathsf{KD}_1(\mathbb{k}_s, m_s) \cdot x_1 + \cdots + \mathsf{KD}_s(\mathbb{k}_s, m_s) \cdot x_s \equiv \mathsf{MAC}_{\mathbb{k}_s}(m_s). \end{cases}$$

Here b *is polynomial in the security parameter* ℓ *and* KD *stands for a key derivation process. If such a solution does not exist, then the* M-MAC *algorithm fails and returns* $\perp$.

(*3*) $v \leftarrow \mathsf{Ver}(\mathbb{k}_i, m_i, \tau)$ *is the verification algorithm that returns* 1 *if and only if having* $\tau = \{x_1, x_2, \ldots, x_s\}$, *it holds that* $\mathsf{MAC}_{\mathbb{k}_i}(m_i) \equiv \mathsf{KD}_1(\mathbb{k}_i, m_i) \cdot x_1 + \mathsf{KD}_2(\mathbb{k}_i, m_i) \cdot x_2 + \cdots + \mathsf{KD}_s(\mathbb{k}_i, m_i) \cdot x_s$. *Otherwise, it returns* 0.

Let us emphasize that the probability that the M-MAC fails to return a solution is negligible in the security parameter (if proper b and s are chosen). As shown in Charlap et al. [1990], the probability that an n-by-n matrix with random elements from $GF(q)$ is nonsingular converges to $\prod_{i=1}^{\infty}(1 - 1/q^i)$ as $n \to \infty$. For example, in cases when $s = 4$, we have a chance for the M-MAC to fail of around 10^{-5} for $b = 16$ and 10^{-10} for $b = 32$.

Example 1. We want to clarify here our intentions on M-MACs with respect to the protocol design. Consider a case when master M broadcasts messages m_1 and m_2 to slaves S_1 and S_2 along with the authentication tag. To increase the efficiency of our protocol, we want to authenticate both messages with the same mixed MAC, and more, since only a portion of each tag is disclosed, for example, 64 bits (reducing the bus overhead but also the security level), we want one of the slaves to be able to carry out the authentication further with a new valid tag (note that this is what happens in the case of the two-stage authentication). Consider that the following packets arrive on the bus: message m_1, message m_2, and the mixed tag obtained by simply concatenating the two tags $\mathsf{MAC}_{k_1}(m_1)||\mathsf{MAC}_{k_2}(m_2)$. However, due to the message filtering feature of the CAN bus, it may be that the two messages do not reach both slaves. Assume message m_1 reaches S_1 and m_2 reaches S_2. Now neither S_1 nor S_2 can carry the authentication further; even in the case when they both have k_1 and k_2, they are not in possession of the message that reached the other slave and thus they cannot validate the other part of the tag. More relevant, note that the nodes are unable to detect if the other part of the tag is compromised. Now consider the case of the LM-MAC. In this case, the tag is obtained by mixing the two tags via the linear equation system; for example, the two components of the tag x_1, x_2 verify a relation of the form $\alpha_1 x_1 + \alpha_2 x_2 = \mathsf{MAC}_{k_1}(m_1)$ and $\beta_1 x_1 + \beta_2 x_2 = \mathsf{MAC}_{k_2}(m_2)$ (here αs and βs are derived from the secret keys k_1, k_2). If an adversary compromises any part of the tag (i.e., either x_1 or x_2), then both equations will fail to verify and any of the receivers possessing a key the adversary has not compromised can detect this (indeed, we assume that the adversary is not in possession of the secret keys k_1 and k_2 since in such case he or she can compute correct LM-MACs anyway). Consequently, with the LM-MACs, any of them can check the tag for correctness and this validation will also hold for the other receiver; this is inherited from the strong nonmalleability property for M-MACs.

For efficiency, we can drop some of the computation from the M-MAC at small penalties in security. The next construction provides such a simplification.

Construction 3 (*Simplified Linearly Mixed Message Authentication Code*). *We define the* SLM-MAC *in the same manner as the* LM-MAC *except for the fact that in the generation and verification algorithms, the message is not used by the key derivation process; that is,* $\mathsf{KD}_i(\mathbb{k}_j, m_j)$ *is replaced by* $\mathsf{KD}_i(\mathbb{k}_j), \forall i, j \in 1..s$.

Both the LM-MAC and SLM-MAC constructions are unforgeable, provided that the underlying MAC is unforgeable. They are also strongly nonmalleable and an adversary cannot manipulate a single element of the tag without making the tag fail on all of the underlying keys (except for negligible probability). As the coefficients are constant in the case of the SLM-MAC, this construction will not provide strong nonmalleability if the adversary learns any of the $\mathsf{MAC}_{\mathbb{k}_i}(m_j)$ values. This would not be the usual case as the authentication tag is composed by $\tau = \{x_1, x_2, \ldots, x_s\}$, from which one cannot build $\mathsf{MAC}_{\mathbb{k}_i}(m_j)$ unless he or she is in possession of $\mathbb{k}_i$ in order to derive the corresponding coefficients $\mathsf{KD}_j(\mathbb{k}_i), j \in 1..s$. We defer the formal treatment of these properties to Appendix A.

2.4. LiBrA-CAN: The Main Scheme

In previous work [Groza et al. 2012], we defined the main authentication scheme around a master-oriented communication. This was justified by the fact that due to the limited size of a standard CAN frame [Robert BOSCH GmbH 1991], one frame would not be enough to carry both the message and the authentication tag. Consequently, using a master node with higher computational power to continue the authentication seemed like a correct practical approach, justified also by the results of the experimental section. However, the master-oriented communication may somewhat conflict with CAN specifications (which clearly specify that the CAN is a multimaster bus), and it also results in more overhead by sending multiple authentication frames (notably, CAN frames have about 50% overhead). Fortunately, as we worked on the protocol, the new CAN standard with flexible data rates, CAN-FD, was released [Robert BOSCH GmbH 2012], and this allows us to place all the authentication information in a single frame, reducing the overhead and making it possible to have a cleaner, crisper protocol specification (we also include results from a real-time simulation of the main scheme on CAN-FD).

NOTATION. To avoid unnecessary formalism that would not impact security, we make some simplifications. Whenever the authentication tag does not fit in a single frame, we assume that it is sent over separate authentication frames, each of them having the proper counter c_{aut} (we do not explicitly use c_{aut} in the description of the schemes since this will only overload the notations). For the same reason, we use the notation $\widetilde{m}$ to denote the message that is already augmented by the counter c_{mes}, identifier id_{frame}, and node identity $\mathcal{N}_i$ (the node identity $\mathcal{N}_i$ can be eventually skipped if it is embedded in the ID field of the frame). Since, with one exception, all versions of the protocol authenticate the same message to all nodes (rather than authenticate a tuple of messages), we replace $\mathbb{M}$ with the augmented message $\widetilde{m}$ and we write $\mathsf{M\text{-}MAC}_{\mathbb{K}}(\widetilde{m})$. Obviously in this case the M-MAC receives as input a message tuple of s identical messages $\widetilde{m}$, that is, $\mathbb{M} = \{\underbrace{\widetilde{m}, \widetilde{m}, \ldots, \widetilde{m}}_{s-times}\}$.

Table III. Possible Groups with 4 Nodes, Groups of Size 2 Outlined

	G_0	G_1	G_2	G_3	G_4	G_5	G_6	G_7	G_8	G_9	G_{10}	G_{11}	G_{12}	G_{13}	G_{14}	G_{15}
$\mathcal{N}_1$									✓	$\boxed{\checkmark}$	$\boxed{\checkmark}$	✓	$\boxed{\checkmark}$	✓	✓	✓
$\mathcal{N}_2$					✓	$\boxed{\checkmark}$	$\boxed{\checkmark}$	✓					$\boxed{\checkmark}$	✓	✓	✓
$\mathcal{N}_3$			✓	✓			✓	✓			✓	✓			✓	✓
$\mathcal{N}_4$		✓		✓		✓		✓		✓		✓		✓		✓

Table IV. Fraction of Uncorrupted Keys on Each Node, in the Case of 8 Participants, with Group Size $g = 1..7$ and Corrupted Nodes $l = 0..8$

g	$\|G\|$	$\|G'\|$	$l = 0$	$l = 1$	$l = 2$	$l = 3$	$l = 4$	$l = 5$	$l = 6$	$l = 7$
1	8	1	12.5%	12.5%	12.5%	12.5%	12.5%	12.5%	12.5%	12.5%
2	28	7	25%	21%	17%	14%	10%	7%	3.5%	–
3	56	21	37.5%	26%	17%	10%	5%	1.7%	–	–
4	70	35	50%	28%	14%	5%	1.4%	–	–	–
5	56	35	62%	26%	8.9%	1.7%	–	–	–	–
6	28	21	75%	21%	3.5%	–	–	–	–	–
7	8	7	87%	12.5%	–	–	–	–	–	–

The key allocation procedure distributes the keys to the n nodes by placing them in groups of size g. Figure 2 provides an example of key sharing for the master-oriented version of the scheme; for the main scheme every sender will be placed in the role of the master. We start with an example that shows how we intend to distribute keys to the nodes, and then we proceed to the main scheme.

Example 2. Table III shows all the 16 groups that can be formed in the case of four nodes, including the empty group G_0 and the group containing all nodes G_{15}. The check mark ✓ is present if the node is part of the group referred to in the column header. We highlight in gray the columns corresponding to all groups that can be formed by two distinct nodes; there are exactly $\binom{4}{2} = 6$ such groups. Moreover, each two distinct nodes share exactly $\binom{2}{0} = 1$ group out of the six. To exemplify further, we outline the groups shared by $\mathcal{N}_1$ (i.e., G_9, G_{10}, G_{12}) and those shared by $\mathcal{N}_2$ (i.e., G_5, G_6, G_{12}) by marking them with a square; it is easy to see that they intersect in a single group (i.e., G_{12}). We now proceed to a broader example. In Table IV, the case of $n = 8$ is explored, with complete groups of all sizes g and any number of corrupted nodes l. We outline the total number of groups $|G|$ and the number of groups shared by each node $|G'|$, as well as the percentage of uncorrupted keys on each node, that is, keys that are not known by the adversary. This value is computed as follows: for $n = 8$ and $g = 2$, there are 28 groups; if there is a single corrupted node (i.e., $l = 1$), the node is part of exactly seven groups and controls $7/28 = 25\%$ of the 28 shared keys. But each honest node shares exactly one key with the dishonest one; consequently, out of the seven keys, only six cannot be controlled by the dishonest node, resulting in a fraction of $6/28 = 21\%$.

Key sharing procedure. There are numerous key-sharing procedures proposed in the literature based on the two well-known and explored alternatives: a secure symmetric-key server (e.g, employed in related work from Hartkopp et al. [2012] and Woo et al. [2015]) and the public key infrastructure (PKI) (e.g., suggested in related work from Wolf et al. [2006]). However, the biggest challenge in the context of in-vehicle security is not in writing some key-exchange protocol but in finding one that can be integrated in real-world deployments where components from a single car come from many distinct manufacturers. Since our work is focused on broadcast authentication rather than key sharing, we believe that integrating a handshake for the key exchange will have little theoretical merit, while a proposal that could ease practical adoption is out of reach for the current work. For this reason, in what follows, we simply assume that the corresponding keys exist on each node. For security reasons, we assume that new keys are negotiated at each protocol reinitialization (e.g., each time the car starts) and refreshed at periodic intervals (e.g., each hour/day depending on the intended security level). We do, however, emphasize that while our implementation frequently uses truncated authentication tags to make them fit in the small CAN data field (a procedure present in any of the related proposals), this does not make the authentication keys vulnerable since the keys can have full lengths (e.g., from 64 to 128 bits). It is just the probability that an adversary can simply guess an authentication tag that is higher due to the truncation. As previously discussed, reinitializing keys is also beneficial to ensure freshness; thus, the size of the message counter can dictate the frequency of reinitialization.

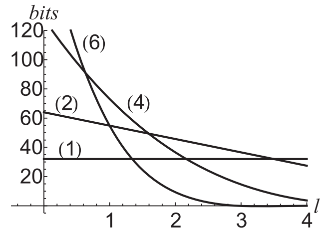
***Construction* 4** (LiBrA-CAN - *Main Scheme*). *Given an* M-MAC *construction for some security parameter ℓ and n nodes placed in groups of size g, we define protocol* LiBrA-CAN$_{\mathcal{N}^*}$(M-MAC, ℓ, n, g) *as the following set of actions for each CAN node denoted as $\mathcal{N}_i, i = 1..n$:*

(*1*) Setup(ℓ, n, g) *is the key setup procedure. Let $t = \binom{n}{g}$ be the number of subsets of g nodes out of the n nodes. The* Setup *procedure generates the t random keys, each of ℓ bits, then distributes to each node the keys for the groups that he or she is part of. Let $\mathbb{K}^i_{\mathcal{N}} = \{\Bbbk^i_1, \Bbbk^i_2, \ldots, \Bbbk^i_{t'}\}$ with $t' = \binom{n-1}{g-1}$ denote the key set of each node $\mathcal{N}_i$, and $\mathbb{K}^{i,j}_{\mathcal{N}} = \{\Bbbk^i_1, \Bbbk^i_2, \ldots, \Bbbk^i_{t''}\}$ with $t'' = \binom{n-2}{g-2}$ the keys shared by each node $\mathcal{N}_i$ with node $\mathcal{N}_j$.*

(*2*) SendMesTag($\mathcal{N}_i, \widetilde{m}, \mathbb{K}^i_{\mathcal{N}}$), *on which node $\mathcal{N}_i$ whenever it wants to broadcast a message $\widetilde{m}$ increments its local counter, computes the tag* M-MAC$_{\mathbb{K}^i_{\mathcal{N}}}(\widetilde{m})$ *with its keyset $\mathbb{K}^i_{\mathcal{N}}$ and sends the message $\widetilde{m}$ and the authentication tag on the bus.*

(*3*) RecMesTag($\mathcal{N}_i, \mathcal{N}_j, \widetilde{m}$, M-MAC$_{\mathbb{K}^i_{\mathcal{N}}}(\widetilde{m})$), *on which node $\mathcal{N}_i$ receives a data frame containing message $\widetilde{m}$ from node $\mathcal{N}_j$ along with the corresponding tag* M-MAC$_{\mathbb{K}^j_{\mathcal{N}}}(\widetilde{m})$. *Node $\mathcal{N}_i$ checks if the message is fresh (i.e., counter up to date) and authentic for all common keys, that is, $1 \leftarrow$* Ver($\widetilde{m}, \Bbbk,$ M-MAC$_{\mathbb{K}^j_{\mathcal{N}}}(\widetilde{m})$), $\forall \Bbbk \in \mathbb{K}^{i,j}$. *If the tag is correct for all keys in the common keyset (i.e., $\mathbb{K}^{i,j}$), then message $\widetilde{m}$ is deemed authentic and the message counter updated.*

Example 3. To see that our scheme achieves superior security, we compare it to the scheme in Szilagyi and Koopman [2010]. We take, for example, a setup with four receivers and one sender. In Szilagy and Koopman's scheme, one frame will be sent containing four MACs (one for each receiver). Set each of these tags to 9 bits (in Szilagyi and Koopman [2010] 8 bits are suggested for each MAC; we use 9 here just to make the comparison fair between the schemes, but these numbers are artificially small anyway and serve just as an example). The security level after the first frame is 9 bits for each node and, regardless of the number of subsequent votes from the other nodes, the security level is at most $(4 - l) \times 9$ bits, where l is the number of corrupted nodes. This is because subsequent votes from other nodes will only confirm that their fragment from the initial MAC is correct and there are only $(4 - l)$ trusted fragments each of 9 bits. Now consider LiBrA in the case of groups of size two and one sender. In this setup, there are six groups and keys shared between the sender and these groups. The sender now has to broadcast six MACs of 6 bits each (this adds up to a 36-bit tag as previously). If there is no corrupted node, as each node has three of the six keys, the security level after the first message from the sender is double compared to Szilagy and Koopman's scheme, that is, $50\% \times 36 = 18$ bits versus 9 bits. Subsequently, each new authentication frame of the sender contributes with the same amount of authentication bits (though our intention is to ensure authentication in a single frame in order to avoid



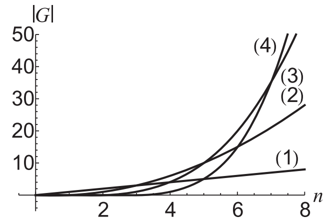


Fig. 4. Fraction of recovered bits (left) for eight nodes with $l = 1..7$ and number of groups (right) for $n = 1..8$ (group size given in the round brackets).

delays and reduce overheads, which can be done simply by increasing the size of each tag). If we set the number of corrupted nodes to one, we still get 12 bits of security after a single message, and the same amount of authentication bits will be added by each new frame (because for each honest node 33% of the key bits are not corrupted). More relevant examples can be drawn by increasing the number of nodes and MAC tags. In the case of eight nodes, as shown later in Table IV, creating groups of size two leads to 25% of security bits being recovered from each tag by each receiver, while increasing the group size to six leads to 75% of security bits (all this in contrast to 12.5% security bits recovered if keys are shared pairwise as in the scheme from Szilagyi and Koopman [2010]). In the case of one corrupted node, the security level is the same for two or six nodes, and in case of two corrupted nodes, it drastically drops but is still higher for groups of size two. Roughly speaking, groups of size two give double or triple security level when there are fewer than two corrupted nodes.

The security of the main scheme can be directly linked to the security of the M-MAC construction that we analyze in Appendix A. Next we give an account for the influence of corrupted principals on the security level (i.e., collusion attacks). We simply point to combinatorial bounds as the security of such multicast schemes is well established; see, for example, Canetti et al. [1999].

Number of groups and keys. For n participants, we have 2^n groups; this includes one empty group and one group that contains all participants. There are $\binom{n}{g}$ possible groups of size g. For n nodes, given all groups of size g, each node is part of $\binom{n-1}{g-1}$ groups. Further, if one considers that there is one corrupted node, then he or she shares with each other node $\binom{n-2}{g-2}$ groups. If there are l corrupted nodes, then they share with each node $\binom{n-l-1}{g-l-1}$ groups (this gives the number of corrupted keys on each node). For a more accurate view, we translate this discussion into security bits. Assume that the M-MAC is built by simple concatenation of regular MACs (each of them computed with the corresponding shared key). Having an M-MAC of some fixed bit length t, each individual MAC is truncated to t/n bits. If keys are pairwisely shared between the sender and receivers (i.e., $g = 1$), then each node receives exactly t/n security bits (since it is in possession of a single key that works for one of the MACs). But in case we share the keys between groups of size g, then each node is in possession of $\binom{n-1}{g-1}$ keys, which is a fraction of $\binom{n-1}{g-1} \cdot \binom{n}{g}^{-1}$ from the total number of keys and equivalently from the bits carried by the M-MAC (obviously this is more than $1/n$ for $g = 1$). Figure 4 shows on the left side the decrease in security bits, having fixed 256 bits for the tag, for groups of size $g = 1, 2, 4, 6$ in the case of 0..4 adversaries (a dishonest minority). If we consider an authentication tag of 256 bits, if this is to be shared by eight nodes with eight different keys, then only $256/8 = 32$ bits per node remain. Contrary, in the case of groups of size $g = 2$, there are 64 bits for each node; even if there is one corrupted node, still 54 bits remain untouched (while this is not much, it may be enough for real-time security). Generally, with one adversarial node, the highest security is for $g = n/2$ due to the binomial expansion in which the middle coefficient is the larger. With at most $n/2$ adversaries (i.e., adversarial minority), the highest security is at $g = 2$ and it decreases linearly. On the right side, Figure 4 shows the number of groups (which translates into keys) in the case of $n = 1..8$ nodes and subgroups of size $g = 1, 2, 3, 4$. Obviously, this grows exponentially, but for smaller g the number of keys is decent and gives strong security benefits.

We further design several variations of the main authentication scheme that give efficient tradeoffs as shown in the experimental results section. These variations can be roughly grouped into two main classes. First, for nonhomogeneous networks, it makes sense to have a *master-oriented authentication* where the node with higher computational power deals with the main part of the authentication procedure. Second, for homogeneous networks where all nodes have similar computational power, it makes sense to have a *distributed authentication* scheme where all participants contribute to the authentication task. For brevity, whenever possible without confusion, we skip the formalism related to the schemes.

2.5. Master-Oriented Versions of the Scheme

A master-oriented communication makes sense since it is practical to have one node with higher computational power that can take care of the most intensive part of the authentication. This is also supported by our experimental results. More, if the master node is a trustworthy third party, there are clear security benefits if he or she handles all keys that are shared between nodes since he or she can continue the authentication further with all the remaining keys (not only with the keys known to the sender). This is summarized by the next construction. Figure 2 shows the master node and the slave nodes connected to the bus; it also outlines the keys that are shared between nodes.

Construction 5 *(Centralized Authentication). Given an* M-MAC *construction for some security parameter ℓ and n nodes placed in groups of size g, we define protocol* $\mathsf{CN\text{-}LiBrA\text{-}CAN}_{\mathcal{M},\mathcal{N}^*}(\mathsf{M\text{-}MAC}, \ell, n, g)$ *as follows. Let $\mathbb{K}_{\mathcal{M}}$ denote the keyset of the master node and run the previous key-setup procedure only for the master node (i.e., the master places the slaves in groups of size g and shares keys with the groups). The following set of actions hold for the master $\mathcal{M}$:*

(*1*) $\mathsf{RecMesTag}(\mathcal{M}, \mathcal{N}_i, \widetilde{m}, \mathsf{M\text{-}MAC}_{\mathbb{K}^i_{\mathcal{N}}}(\widetilde{m}))$, *on which master $\mathcal{M}$ receives message $\widetilde{m}$ and authentication of the tag* $\mathsf{M\text{-}MAC}_{\mathbb{K}^i_{\mathcal{N}}}(\widetilde{m})$ *from slave $\mathcal{N}_i$. Subsequently, master $\mathcal{M}$ checks if the counter is up to date and if the message is authentic, that is,* $1 \leftarrow \mathsf{Ver}(\widetilde{m}, \Bbbk, \mathsf{M\text{-}MAC}_{\mathbb{K}^i_{\mathcal{N}}}(\widetilde{m}))$, $\forall \Bbbk \in \mathbb{K}^i_{\mathcal{N}}$. *If so, he or she proceeds to authenticating the tag to other nodes with* $\mathsf{SendTag}(\mathcal{M}, \widetilde{m}, \mathbb{K}^i_{\mathcal{N}})$ *and updates the message counter.*

(*2*) $\mathsf{SendTag}(\mathcal{M}, \widetilde{m}, \mathbb{K}^i_{\mathcal{N}})$, *on which master $\mathcal{M}$ gathers all the remaining keys* $\mathbb{K}_{\mathcal{M}} \setminus \mathbb{K}^i_{\mathcal{N}}$, *computes* $\mathsf{M\text{-}MAC}_{\mathbb{K}_{\mathcal{M}} \setminus \mathbb{K}^i_{\mathcal{N}}}(\widetilde{m})$, *and broadcasts it as an authentication frame with the same ID as the original message (note that in this case the* M-MAC *is computed with the remaining* $\binom{n}{g} - g$ *keys; there is no restriction from the construction of* M-MAC *to do so).*

The following set of actions hold for each of the slaves $\mathcal{N}^*$:

(*1*) $\mathsf{RecMesTag}(\mathcal{N}_i, \mathcal{N}_j, \widetilde{m}, \mathsf{M\text{-}MAC}_{\mathbb{K}^j_{\mathcal{N}}}(\widetilde{m}))$, *on which slave $\mathcal{N}_i$ receives message $\widetilde{m}$ and an authentication tag from another slave $\mathcal{N}_j$ and proceeds similarly to master $\mathcal{M}$ by checking if the message is authentic but only with respect to the keys* $\Bbbk \in \mathbb{K}^i_{\mathcal{N}} \cap \mathbb{K}^j_{\mathcal{N}}$

that he or she shares with slave $\mathcal{N}_j$. The message is not deemed authentic until a successful $\mathsf{RecTag}(\mathcal{N}_i, \mathcal{M}, \mathcal{N}_j, \mathsf{M\text{-}MAC}_{\mathbb{K}_\mathcal{M} \setminus \mathbb{K}^j_\mathcal{N}}(\widetilde{m}))$ *event follows.*

(2) $\mathsf{RecTag}(\mathcal{N}_i, \mathcal{M}, \mathcal{N}_j, \mathsf{M\text{-}MAC}_{\mathbb{K}_\mathcal{M} \setminus \mathbb{K}^j_\mathcal{N}}(\widetilde{m}))$, *on which slave $\mathcal{N}_i$ receives an authentication frame containing the tag* $\mathsf{M\text{-}MAC}_{\mathbb{K}_\mathcal{M} \setminus \mathbb{K}^j_\mathcal{N}}(\widetilde{m})$ *from master $\mathcal{M}$ (that continues the authentication of slave $\mathcal{N}_j$) and verifies for all keys $\Bbbk \in \mathbb{K}^i_\mathcal{N} \cap (\mathbb{K}_\mathcal{M} \setminus \mathbb{K}^j_\mathcal{N})$ if the tag is correct. If for all keys in its keyset the tag is correct, then message $\widetilde{m}$ is deemed authentic and the message counter updated.*

(3) $\mathsf{SendMes}(\mathcal{N}_i, m, \mathbb{K}^i_\mathcal{N})$, *on which slave $\mathcal{N}_i$ whenever he or she wants to broadcast a message $\widetilde{m}$ increments its local counter, computes the tag* $\mathsf{M\text{-}MAC}_{\mathbb{K}^i_\mathcal{N}}(\widetilde{m})$ *with its keyset $\mathbb{K}^i_\mathcal{N}$, and sends the data frame containing message $\widetilde{m}$ and the corresponding tag on the bus.*

Cumulative authentication. Since in some scenarios small delays may be acceptable, we can take benefit of them and increase the efficiency of the main scheme. In the *cumulative authentication* scheme, a timer can be used and all messages are accumulated by the master over a predefined period δ and then authenticated at once (this procedure can be employed in the slave-only settings as well). While this introduces an additional delay δ, similar to the case of the TESLA protocol, this delay can be chosen as small as needed to cover application requirements. Distinct to the case of the TESLA protocol, the delay is not strongly constrained by external parameters (such as oscillator precision, synchronization error, bus speed, etc.).

Load-Balanced Authentication. The *centralized authentication* scheme is beneficial in the case when the communication master has higher computational resources, but it may be the case that the master node is already busy with other computational tasks. For such case, a *load-balanced* version of the scheme can be used in which the communication master can send a flag (authenticated along the message) to a point for a particular slave to carry the authentication further.

2.6. Distributed Versions of the Scheme

Indeed, an authentication master may not always be present. Moreover, several events can lead to his or her unavailability (e.g., he or she can enter bus off-mode due to problems with the transceiver or the ECU itself can suffer a malfunction). For this purpose, we introduce the cascade authentication scheme where the slaves reply in a cascade manner by sending the authentication tag for their group of keys. In what follows, we assume that all the nodes continue the authentication in a round-robin fashion until they reach the sender (or stop after sufficient authentication tags are released). Thus, we point to node $(i+1)$ as the next node, and whenever we reach the nth node, the first one becomes the next, and so forth.

Construction 6 *(Cascade Authentication). Given an* $\mathsf{M\text{-}MAC}$ *construction for some security parameter ℓ and n nodes placed in groups of size g, we define protocol* $\mathsf{DC\text{-}LiBrA\text{-}CAN}_{\mathcal{N}^*}(\mathsf{M\text{-}MAC}, \ell, n, g)$ *as follows. Let $\mathbb{K}^{i,j}_\mathcal{N}$ denote the keys shared between nodes i and j (we assume the same key-setup as previously, except that the master does not play any role in authentication). Then the following set of actions is defined for each of the nodes $\mathcal{N}^*$:*

(1) $\mathsf{RecMes}(\mathcal{N}_i, \mathcal{N}_j, \widetilde{m})$, *on which node $\mathcal{N}_i$ receives a data frame containing message $\widetilde{m}$ from another node $\mathcal{N}_j$ and checks if the counter is up to date, then stores the message in a queue of messages to be authenticated.*

(2) $\mathsf{RecTag}(\mathcal{N}_i, \mathcal{N}_j, \mathsf{M\text{-}MAC}_{\mathbb{K}^{j,j+1}_\mathcal{N}}(\widetilde{m}))$, *on which $\mathcal{N}_i$ receives an authentication frame containing tag* $\mathsf{M\text{-}MAC}_{\mathbb{K}^{j,j+1}_\mathcal{N}}(\widetilde{m})$ *from another node $\mathcal{N}_j$ and verifies for all keys*

$\Bbbk \in \mathbb{K}^{i}_{\mathcal{N}} \cap \mathbb{K}^{j,j+1}_{\mathcal{N}}$ *if the tag is correct. If for all keys in its keyset a correct tag was received, then message* $\widetilde{m}$ *is deemed authentic and the message counter updated. If* $i = j + 1$, *then it proceeds to* $\mathsf{SendTag}(\mathcal{N}_i, \widetilde{m}, \mathbb{K}^i)$.

(3) $\mathsf{SendTag}(\mathcal{N}_i, \widetilde{m}, \mathbb{K}^i)$, *on which node* $\mathcal{N}_i$ *gathers all the keys shared with node* $\mathcal{N}_{i+1}$ *in the set* $\mathbb{K}^{i,i+1}_{\mathcal{N}}$, *computes* $\mathsf{M\text{-}MAC}_{\mathbb{K}^{i,i+1}_{\mathcal{N}}}(\widetilde{m})$, *and broadcasts it as an authentication frame.*

(4) $\mathsf{SendMes}(\mathcal{N}_i, \widetilde{m}, \mathbb{K}_i)$, *on which node* $\mathcal{N}_i$ *whenever it wants to broadcast a message* $\widetilde{m}$ *increments its local counter, computes the tag* $\mathsf{M\text{-}MAC}_{\mathbb{K}^{i,i+1}_{\mathcal{N}}}(\widetilde{m})$, *and sends the data frame containing* $\widetilde{m}$ *followed by an authentication frame containing the tag on the bus.*

SECURITY. The cascade authentication is intended for balancing the computational costs of the authentication between principals. Since the nodes proceed in a chain reaction by simply reauthenticating messages with their own keys, new tags are produced that extend the authentication over new keys. The security level at each node is at most that of the tag from the initial message that triggered the cascade (assuming all nodes in the cascade are trusted; otherwise, this is proportional to the number of uncorrupted keys). Consequently, the security level depends solely on the size of the individual tags that are used for the response of the honest receivers that continue the cascade (by further providing correct authentication tags with the correct keys that they hold). For this reason, we advise that a sufficient security level must be chosen to avoid an exhaustive search of the correct authentication tag by the injection of faked messages until an honest node will continue the cascade. For the LM-MAC construction, while the size of the tag remains the same and it requires the same number of attempts to obtain a valid response from an individual node, it is not possible to attack small fragments of the tag (by exhaustive search on each fragment until a node provides a valid response) and later recompose the larger tag from the smaller fragments (e.g., by simple concatenation). Thus, the LM-MAC provides additional security benefits in front of simple concatenation. We do not further explore these benefits for the cascade authentication scheme since we believe that for practical needs the main LiBrA-CAN scheme is the optimal choice as tags are sent at once with the messages, avoiding supplementary bus load and facilitating immediate authentication.

TWO-STAGE AUTHENTICATION. In the case of *two-stage authentication*, we assume a scenario with nodes of equal computational power. In this case, each node can start broadcasting by sending a tag that includes only a part of the keys for the subgroups that he or she is part of and a second node (pointed out by some flag, or predefined in protocol actions) continues with the authentication. The procedure is repeated until the desired number of authentication frames is reached.

MULTIMASTER AUTHENTICATION. For the same reasons, a distributed version of the *centralized authentication* scheme can be imagined. In this case, several nodes with higher computational power can form a group of communication masters. Each of them may broadcast a distinct authentication tag, and if any such tag is missing due to the unavailability of a particular node, the other masters will take care of replacing this tag with one of their own.

3. EXPERIMENTAL RESULTS

To evaluate the performance of the proposed protocol suite, we used several setups with different hardware components, with the goal to determine the minimum authentication delay.

Table V. Technical Specifications for S12XDT512 and TriCore TC1797

Characteristic	S12XDT512	TriCore TC1797
Flash Memory	512KB	4MB
RAM	20KB	156KB (SRAM)
EEPROM	4KB	16KB emulated by 64KB data Flash
Clock Speed	80MHz	180MHz

3.1. Protocol Performance

Automotive-grade embedded devices from Freescale and Infineon as well as a notebook equipped with an adapter for CAN communication from Vector were employed. On the S12X platform, only the main core was used at a frequency of 80MHz (the X stands for the XGATE coprocessor that can be further used to optimize computations). The embedded platforms that we used are representative for industry's low-end and high-end edges. Table V provides several technical details on these two platforms. We built several testbeds as follows:

- Testbed T1: $\mathit{Intel}\,T7700{+}4\times TC1797$. The master node is implemented on a standard PC connected via a high-speed CANcab connector from the CANcardXL board to the slave nodes that are built on the TriCore platform (this enables a 1Mbps communication speed).
- Testbed T2: $TC1782{+}4\times TC1797$. Master and slave nodes are built on similar TriCore (TC1797) development boards. The CAN communication speed is set to 1Mbps.
- Testbed T3: $\mathit{Intel}\,T7700{+}4\times S12X$. The master node is implemented on a PC (Intel Core2Duo CPU T7700@2.4GHz), while slave nodes are built on the S12X boards. The master-slave CAN communication is done through the CANcardXL using a low-speed CANcab for 125Kbps.
- Testbed T4: $S12X{+}4\times S12X$. Both master and slave nodes are built on identical S12X development boards with CAN communication speed set to 125Kbps.
- Testbed T5: $S12X{+}8\times S12X$. We used eight nodes based on S12X boards for cascade and two-stage authentications. To these, one more S12X node was added as master for the case of centralized authentication.

Centralized authentication was tested for the case of four nodes in groups of size two, which leads to a total of six groups. Messages and authentication tags are always sent as separate frames and the message size is always 8 bits, for example, the size of a message from an analog-to-digital converter. The MAC size for each key is truncated to 21 bits so that three authentication tags fit a single 64-bit CAN frame. The MAC is computed using the MD5 hash function over an input formed by concatenating the group key to the message. While we are aware that MD5 is weak and collisions were found long ago, we use it as a baseline for speed comparisons. Since real-time attacks were not reported so far on MD5, it should still be secure for the setting that we address. For completeness and to serve as a thorough comparison, we will also provide results for other relevant hash functions, including the SHA3 candidates in Table VIII. The resulting hash is then truncated to the desired size. Table VII holds the timings and bus loads for each testbed. Here δ is the authentication delay, that is, the delay between the time at which a message is received up to when it is authenticated on the receiver side (this includes the time to verify the tag and partly the time to compute and send it since these two can start as soon as the message is sent). For the bus load, we considered the fraction of traffic caused by the authentication tags over the entire bandwidth.

As expected, scenarios in which high-end devices played the role of master nodes (PC, TriCore) showed better performances than in the case of low-end master nodes. The case

Table VI. Example of Tag Allocation for the Cascade Scheme DC-8S2F4

-	$\mathcal{N}_1$	$\mathcal{N}_2$	$\mathcal{N}_3$	$\mathcal{N}_4$	$\mathcal{N}_5$	$\mathcal{N}_6$	$\mathcal{N}_7$	$\mathcal{N}_8$
$\mathcal{N}_1$	–	36	10	10	8	–	–	–
$\mathcal{N}_2$	–	–	26	10	10	18	–	–
$\mathcal{N}_3$	–	–	–	18	10	10	26	–
$\mathcal{N}_4$	–	–	–	–	8	10	10	36

of a PC master with TriCore slaves does not perform better, despite the considerable difference in computational power between master nodes (TriCore vs. Intel Core2Duo) due to limitations of CAN adapters. Because of their internal hardware/software design, these adapters introduce some limitations (e.g., the average response time specified by Vector for the CANcardXL is $100\mu s$).

A more complex setting was implemented around eight S12X nodes grouped two by two, which leads to 28 groups. The size of the authentication tags and the truncated MAC size differs in each variant. We set up the implementations as follows:

- *Centralized:* The sender computes and sends one MAC for each group that he or she is part of. The master computes and sends one MAC for each of the other 21 groups (if groups of size two are used). If the master is to perform the authentication in only two frames, then each MAC can be truncated to 5 bits and this will lead to a total of 35 security bits for each node. But if we increase the number of authentication frames from the master to three, then each MAC can be truncated to 9 bits, giving a total of 63 authentication bits for each node, which is a reasonable level for real-time security.
- *Cascade:* Three additional nodes take part in the authentication process besides the sender. The sender computes four MACs, three of which are for the nodes that will help to authenticate the message. The helper nodes will then compute three extra authentication tags to provide enough authentication information for all other nodes. An example of tag allocation is suggested in Table VI. On each line, 64 bits are distributed to the nodes denoted on the columns (this is achieved by concatenating MACs that are truncated to the corresponding bit length). Here $\mathcal{N}_1$ is the sender and nodes $\mathcal{N}_2$, $\mathcal{N}_3$, and $\mathcal{N}_4$ continue the authentication. If the size of the groups is two, then each node will get around 36 security bits, but if the size of the groups is increased to three, then, at the same computational cost and bandwidth, around 63 security bits are received by each node.
- *Two-stage:* The master node is missing in this implementation; therefore, we use two helper nodes for computing and sending the complete authentication tag. In the two-stage variant, the sender will first put one authentication tag on the bus that contains the full 36 authentication bits for one of the helper nodes, 20 bits for the second one, and 8 extra bits for another node. This first tag is followed by a second tag generated by the first helper node, which contains the remaining 16 authentication bits for the second helper node and 48 bits equally distributed for three of the remaining nodes. To complete the 36 authentication bits for each of the remaining nodes, the sender and the second helper node will each put an authentication tag on the bus. As discussed previously, the security level can be raised to around 64 bits by using groups of size three and the described tag allocation procedure.

Table VII holds the results achieved with these three implementations. The worst performer in terms of authentication delay is the implementation of the centralized authentication variant as it involves computing MACs for each of the 28 groups in a sequential manner. In the other implementations, a smaller number of MACs are computed, some of which are done by different nodes in parallel. A smaller authentication

Table VII. Authentication Delay and Bus Load (Various Configurations)

Setup and Scheme	Tag (Bits)	Msg. (Bits)	δ (ms)	Bus Load	Speed
T1: Centralized n=4, g=2	64	8	0.267	54.31%	1Mbps
T2: Centralized n=4, g=2	64	8	0.378	42.54%	1Mbps
T3: Centralized n=4, g=2	64	8	2.54	53.84%	125Kbps
T4: Centralized n=4, g=2	64	8	1.848	72.22%	125Kbps
T5: Centralized n=8, g=2	64	8	22.62	11.27%	125Kbps
T5: Cascade n=8, g=2	64	8	9.86	36.11%	125Kbps
T5: TwoStage n=8, g=2	64	8	6.806	46.21%	125Kbps

Table VIII. Execution Time for Some Common Hash Functions

Function	Input Size (Bytes) S12X 0	S12X 16	S12X 64	TriCore 0	TriCore 16	TriCore 64
MD5	$371\mu s$	$374\mu s$	$689\mu s$	$10.16\mu s$	$11.00\mu s$	$18.34\mu s$
SHA1	$1.144ms$	$1.148ms$	$2.285ms$	$14.64\mu s$	$15.10\mu s$	$27.60\mu s$
SHA256	$2.755ms$	$2.755ms$	$5.440ms$	$41.70\mu s$	$42.35\mu s$	$80.80\mu s$
BLAKE 32	$1.496ms$	$1.532ms$	$2.925ms$	$29.78\mu s$	$28.67\mu s$	$48.64\mu s$
BLAKE-2 32	$1.117ms$	$1.119ms$	$1.126ms$	$27.97\mu s$	$28.13\mu s$	$28.13\mu s$
Groestl 256	$2.990ms$	$2.991ms$	$4.919ms$	$237\mu s$	$222.8\mu s$	$359.2\mu s$
JH 256	$6.187ms$	$12.33ms$	$12.30ms$	$104.1\mu s$	$191.5\mu s$	$191.1\mu s$
Keccak	$7.649ms$	$7.654ms$	$7.660ms$	$98.44\mu s$	$98.41\mu s$	$98.4\mu s$
Skein	$5.402ms$	$5.405ms$	$5.410ms$	$183.5\mu s$	$183.4\mu s$	$183.4\mu s$

delay is obtained when using the two-stage implementation at the cost of an increased CPU load on the sender side.

3.2. Computational Performance with LM-MAC

Previous results were based on the simple concatenation of individual MACs computed with MD5 as the underlying hash function. We now take a brief account of the impact of mixing tags using linear systems of equations.

In Table VIII, we give an overview on the computational timings for various hash functions and input sizes on both of the employed platforms. The usual message size for our scenario will be less than 64 bits (the maximum size of the data carried in one CAN frame) since it contains values from various sensors and so forth that are usually small. To this, one will need to add the size of the key that is hashed with the message; thus, 16 bytes for the input should be the expected length. We also give measurements for a 64-byte input just to get an upper bound since an input of such size is less likely (this is in fact the maximum size of the CAN-FD data field). For the Linearly Mixed MACs, in addition to the computation of the MACs, two additional computational tasks are required: solving the linear system of equations on the sender side (a task that should usually be done by the master, which has higher computational power) and reconstructing the MAC on the receiver side.

We made a customized implementation dedicated for $GF(2^{16})$ and $GF(2^{32})$, which resulted in a less general but more compact source code without any unnecessary operation (note that more than a single authentication frame is sent, and thus security is not limited to a single $GF(2^{32})$ part of a tag). We improved more by noticing that we could perform the same mixing procedure by working in the integer group Z_p, where p is Mersenne prime, that is, a prime of the form $2^q - 1$ for some other prime q, since in this case modular reduction can be performed more efficient. Concretely, for a group size close to $GF(2^{32})$, we chose $p = 2^{31} - 1 = 2,147,483,647$, which allows 31 bits of entropy for any of the mixed tags. Having this fixed, the computational

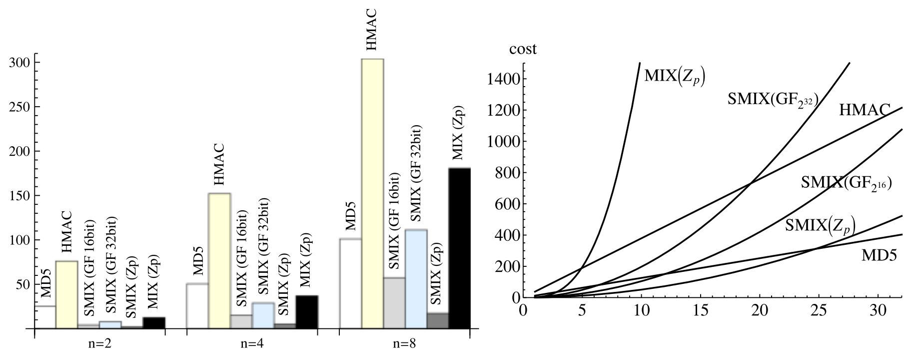


Fig. 5. Computational costs (Infineon TriCore) for mixing the tags (left) and estimation for the influence of the participants' number on computational costs (right).

Table IX. Execution Time for MAC Mixing (Infineon TriCore)

	SMIX (GF16)	SMIX (GF32)	SMIX (Z_p)	MIX (Z_p)
2×2	4.205μs	7.89μs	2.045μs	12.50μs
4×4	15.16μs	28.80μs	5.17μs	36.95μs
8×8	57.3μs	111.0μs	17.28μs	180.20μs

performance was up to about one order of magnitude cheaper than in the case of the $GF(2^{32})$ field. In Figure 5(i), we make a graphical depiction of the computational costs for the case of $n = 2$, 4, and 8 nodes, respectively. In each case, the computational costs of mixing the MACs is compared to the computational cost of two, four, and eight MD5 or HMAC-MD5 operations (these are required to compute the authentication tags in any case). The simplified mixing is significantly cheaper compared to the MD5; in the case of Z_p, its cost is relatively inexpensive compared to the required hashes. Even in the case of the linear mixing based on Gaussian elimination in Z_p, it is still half the cost of the corresponding hashes. Table IX shows the computational costs of simplified linearly mixing SMIX (based on matrix multiplication and required for the SLM-MAC) in the case of $GF(2^{16})$, $GF(2^{32})$, and $Z_{2^{31}-1}$ and the case of regular mixing MIX (based on Gaussian elimination and required by LM-MAC) in the case of $Z_{2^{31}-1}$. In Figure 5(i), we make a graphical representation of the costs when compared to the costs of regular MD5s or HMACs required for the corresponding number of nodes. Figure 5(ii) shows a prediction of the cost increase when comparing the symmetric functions with the additional costs for mixing the tags. The prediction is based on the fact that mixing with linear equations (MIX) is done by Gaussian elimination, which is $O(n^3)$, while matrix multiplication by a vector (SMIX) required for simplified mixing is $O(n^2)$. Finally, the number of MD5s or HMACs increases linearly with the number of nodes. Even in the case of $n = 32$ nodes, the cost for mixing the tags is lower than the cost incurred by the corresponding HMACs.

4. IMPROVEMENTS WITH CAN-FD AND CAN+

In this section, we discuss two methods for eliminating the drawbacks that arise from the limited size of the data field from standard CAN frames, first by employing the recently released CAN-FD standard and second by using an unofficial extension called CAN+ [Ziermann et al. 2009].

4.1. CAN with Flexible Data-Rate (CAN-FD)

Since CAN-FD-capable boards are not yet widely available on the market, we take advantage of the CANoe tool from Vector (www.vector.com), which is the industry

Table X. Bus Load at Immediate Authentication (Main Scheme, CANoe Simulation)

Scheme	Tag (Bits)	Msg. (Bits)	Cycle (ms)	Bus Load CAN	Bus Load CAN-FD	Bus Load (No Auth) CAN	Bus Load (No Auth) CAN-FD
Main Scheme n=4, g=2	64	8	1	67.71%	17.74%	21.76%	13.12%
Main Scheme n=4, g=2	64	64	10	9.85%	2.07%	4.68%	1.6%
Main Scheme n=4, g=2	64	64	1	92.4%	19.61%	46.6%	16.12%
Main Scheme n=8, g=2	64	64	10	19.14%	4.06%	9.36%	3.2%
Main Scheme n=4, g=2	192	64	1	100%	26.26%	46.6%	16.12%
Main Scheme n=4, g=2	192	64	10	19.41%	2.86%	4.68%	1.6%
Main Scheme n=8, g=2	448	64	1	100%	33.77%	93.2%	32.24%
Main Scheme n=8, g=2	448	64	10	37.97%	4.29%	9.36%	3.2%

standard for developing real-time simulations of in-vehicle networks. We underline that this is a real-time simulation and the software allows connection with real-world networks via various hardware interfaces (e.g., VN1630).

Our simulation considers networks of four or eight nodes in groups of size two that send messages in a cyclic manner at 1 or 10ms. The messages are always 8 bytes long (the maximum allowed in CAN frames) followed by the authentication tag (which shares the same frame with the message in case of CAN-FD). With CAN-FD, the LiBrA Main Scheme can deliver immediate authentication since the MAC tag can be sent with the message in the generous 64-byte data field. Thus, there is no authentication delay in our simulation (in contrast to previous experiments from Table VII) since this delay depends solely on the computational power of the node (for which results in Tables VIII and IX should be taken into account). To emphasize the benefits of CAN-FD, we compare the bus load with that of CAN in the case when nodes are broadcasting messages at 1 or 10ms cycles. For the CAN simulation, every node sends a message and an additional authentication tag (i.e., two frames) at each cycle (immediate authentication is not possible on CAN).

Table X illustrates the simulation parameters and the bus loads obtained for both CAN (running at 1Mbaud) and CAN-FD (running at 1Mbaud with the data segment at 8Mbaud). The first lines show the behavior for 64-bit tags at various number of nodes, group sizes, and cycles; CAN copes with all of these, but the bus load for CAN-FD is at least 4$\times$ lower. We also increased the size of the authentication tag to 192 and 448 bits, respectively. This allows a very high security level (e.g., in case no corrupted nodes are present); each node harvests 50% of security bits for $n = 4$ and 12.5% for $n = 8$. CAN-FD behaves perfectly well with bus loads under 25% to 33%. Note that in the case of standard CAN, there is a 100% bus load and some nodes do not manage to transmit messages. In particular, for the case of a 192-bit tag, only two CAN nodes can send messages along with all the authentication tags, while at 448 bits, there is only one node that succeeds so. In contrast to Table VII, since for CAN-FD we don't have separate frames that carry the authentication tag, the bus load covers the entire traffic. Out of this, the authentication tag represents less than 1/2 for 64-bit tags and is sent in the high data rate segment, which works 8$\times$ faster than usual (8Mbps vs. 1Mbps). This leads to a bus load of only 33% even in the case of the larger 488-bit tags.

4.2. Backward Compatibility with CAN+

The CAN+ protocol allows transmission of extra data along with a CAN packet on an out-of-band channel. It does this by transmitting data at an increased rate in between CAN sample points. At least 225 extra bits can be transmitted with the CAN+ protocol alongside a CAN message.

Using the CAN+ protocol for LiBrA-CAN data transmission helps in two ways. First of all, the required bandwidth drops. For LiBrA-CAN schemes, whereby a single node

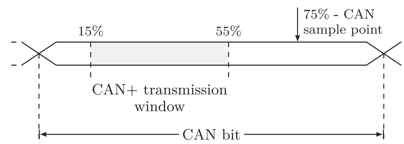


Fig. 6. Timeframe during which the CAN+ protocol can insert extra data into a CAN bit.

never needs to transmit more than $\lceil \frac{225}{64} \rceil = 3$ authentication tags, all LiBrA-CAN data can be transmitted as CAN+ data. This reduces the LiBrA-CAN overhead for those schemes to 0%. Nodes that need to transmit just a tag can do so by transmitting a 0-byte CAN message and embedding the tag as CAN+ data, thereby reducing the time they use the bus from 108-bit lengths (for an 8-byte message) to 44-bit lengths in nonextended CAN mode, which is a 60% decrease.

Second, if LiBrA-CAN authentication data is only transmitted as CAN+ data, then nodes that do not support CAN+ will not even see the LiBrA-CAN data. Thus, a system can be set up whereby important nodes are outfitted with a CAN+ transceiver, while nonimportant nodes aren't. This makes the LiBrA-CAN protocol completely backward compatible with existing CAN networks: nodes supporting CAN+ could be dropped into the network at will and start authentication messages with LiBrA-CAN, while existing CAN nodes will be completely oblivious as to what is going on and continue functioning as before. An added bonus is that this also drastically reduces roll-out cost.

The CAN+ protocol [Ziermann et al. 2009] allows out-of-band data transmission by inserting extra data into the CAN bitstream in between CAN sample points. To better understand this, one must first take a step back and take a look at the CAN protocol.

At the data link level, CAN works with a nondestructive arbitration mechanism. The first step of message transmission consists of sending the message's ID. Due to synchronization, all nodes on a bus will start transmission at the same time. If one of the nodes transmits a 0 (dominant) bit, the bus will be pulled low, thereby "destroying" the transmission of any one (recessive) bit being transmitted at the same time. Stopping transmission as soon as such a collision is detected leads to the nondestructive arbitration mechanism, which prioritizes low-value IDs. One of the requirements of the arbitration processes is that transceivers detecting a collision do not transmit any more bits, so they need to be able to detect a collision on a bit in the same time window during which that bit is sent. This requires the propagation delay to be less than one clock cycle. However, after the arbitration step, only a single node will still be transmitting, and thus, there is no more need to detect bit errors during their transmission window, meaning transmission can happen at an increased rate.

The idea to split CAN message transmission in a slow and fast phase was first proposed by Cena and Valenzano [1999]. The CAN+ protocol takes a slightly different approach: CAN messages themselves are transmitted at their usual rate; however, in between sample points, extra data is sent at an increased rate. A traditional CAN transceiver will not notice these extra bits, thereby making the CAN+ protocol function effectively out-of-band. Figure 6 shows the transmission window for CAN+ data inside a CAN bit. Up to 15 extra CAN+ bits can be transmitted along with each CAN bit on a 1MHz bus. On slower buses, even more bits can be inserted into the data stream. If we assume that CAN+ bits are inserted during the data and CRC field transmission, then a minimum of 225 CAN+ bits can be transmitted (inside the 15 CRC bits, for a 0-byte message on a 1MHz bus).

We implemented a CAN+ transceiver to get a better idea of its area requirements. The basis of this implementation is revision 1.47 of the open-source CAN transceiver implementation by Igor Mohor, found on the OpenCores website (http://www.opencores.org).

Table XI. Synthesis Results for CAN & CAN+ Transceiver on a Spartan-6 Using Xilinx ISE 12.2 (Goal: "Area Reduction")

	Slices	Registers	LUTs
CAN	499	603	917
CAN+	106	95	227
Combined	430	698	1,192

The design was synthesized with Xilinx ISE 12.2 for a Spartan-6 FPGA with design goal "area reduction." Our design takes into account the worst-case maximum CAN+ data length of 225 bits and, for the given synthesis results, cannot transmit or receive more than that. The required area can be found in Table XI. Since it is able to piggyback onto the existing CAN interface logic, the CAN+ transceiver only increases the total transceiver size by a moderate 20%, for a total size of only 430 slices. The total size of the design is smaller than the sum of the size of the two subdesigns (CAN and CAN+), since the Xilinx ISE tool executes compacting operations on the full design. A minimum clock speed of 80MHz is required for the CAN+ transceiver to function on a 1MHz CAN bus. Our design meets those requirements with a maximum clock speed of 90.75MHz.

4.3. Final Words on Performance and Comparison to Related Approaches

With respect to bandwidth, the CANoe simulation clearly proves that the bus load easily reaches 100% in the case of CAN, while it always stays within reasonable margins with CAN-FD. This result is not unexpected and clearly indicates that CAN-FD should be the layer to deploy security, while for the basic CAN, security is achievable only for a limited number of nodes (or messages). We can of course always improve on this by limiting the size of the authentication tag and avoiding to authenticate all frames (e.g., as proposed in Hartkopp et al. [2012]), but this will result in a bus that is only partially secure (an unusual choice as we usually design protocols with the intention to be fully secure). CAN-FD seems the ideal layer for LiBrA-CAN; indeed, it will help all other proposals as well and not only ours in particular.

From a computational point of view, the high-grade Infineon TriCore easily handles the hash computations required by our protocol. The computational requirements do not appear to be problematic as bandwidth appears to be the limiting factor. Related work also considered the use of encryption in counter mode to ensure the confidentiality of frames (e.g., Woo et al. [2015]). While encryption adds an extra layer of security, there are not many advantages in keeping the information secret; in fact, this could make other tasks such as bus monitoring and debugging more difficult. The use of CMAC based on AES rather than hash-based MAC was considered in Hartkopp et al. [2012]. Our Mixed MACs could also be built on top of the AES-based CBC-MAC, and this would have been the construction of choice for us if AES hardware support existed. We believe that the final word on the exact cryptographic primitive to be used would come from the existing hardware support, likely AES or the newer SHA3, that will clearly be available on future-generation automotive grade controllers. Mixing the authentication tags adds some overhead, but it does not appear to be significant.

The experimental results that we provide are meant to prove that implementing the protocol is feasible and to give an image over the computational costs and bandwidth. Comparing the performance of our protocol with distinct proposals cannot be done in a straightforward manner. In Table II of the introductory section, we did provide a comparative overview of the strengths and weaknesses for current proposals. However, a quantitative performance comparison with related work is not really feasible as the goals and design intentions behind the proposals are quite distinct. For example, it is not really possible to compare a TELSA-like protocol (e.g., Groza and Murvay [2013]) with protocols based on pairwise shared keys because obviously the first needs extra bandwidth to broadcast the key chains, even if there are no messages, and it induces authentication delays, while pairwise key sharing is not feasible when the number of nodes is high. A solution such as CaCAN [Kurachi et al. 2014] has almost nonexistent bandwidth requirements and is fully backward compatible, but if the centralized node used for authentication is unplugged from the bus, then all security is lost. Again, a comparison will be useless. Mutatis mutandis, all other solutions based on pairwise key sharing such as Hartkopp et al. [2012], Wang and Sawhney [2014], Bittl [2014], Woo et al. [2015], and LiBrA would revolve around the same performances if the proper parameters are tuned (i.e., group size, number of nodes under the same key, and the size of the authentication tag). Still, the group key sharing behind LiBrA is a mechanism that was recognized long ago to have advantages when adversaries are in the minority (e.g., Fiat and Naor [1994] and Canetti et al. [1999]), and we believe that LiBrA is a better alternative if in-vehicle networks fit into this scenario.

5. CONCLUSION

LiBrA-CAN is an efficient alternative to achieve source authentication if nodes can be placed in small broadcast groups with dishonest nodes in the minority. We expect this to be the case in many automotive scenarios where, although the number of ECUs may be high, the numbers of manufacturers from which they come may not be high and distributing trust between several groups is an acceptable solution. Experiments performed on the recently released CAN-FD showed that it is possible to achieve immediate authentication at small costs in bandwidth. If the number of nodes is high, we see only two resolutions: public key cryptography (with the drawback of high computational requirements, at least two orders of magnitude) or TESLA-like protocols (with the drawback of authentication delays as shown in Groza and Murvay [2013]). CANAuth [Van Herrewege et al. 2011] is also a solution for a high number of nodes if one considers that source authentication is not relevant and associating keys to message groups is sound from a security perspective. While a decision on real-world protocol deployments can be taken only by manufacturers and by means of consortium, we believe that LiBrA carries key concepts for such a deployment. We also consider that the proposal here has the advantage of being simple to implement, and simplicity is one relevant criterion for adoption in practice.

APPENDIX

A. SECURITY PROOFS

We now give a formal account of the properties that we require for the M-MAC constructions. These are proved in a framework of two attack games: one for the unforgeability of the M-MAC (i.e., $\mathsf{Game}^{\mathsf{UF}}_{\mathsf{M\text{-}MAC}}$) and the other for the strong nonmalleability of the M-MAC (i.e., $\mathsf{Game}^{\mathsf{SNM}}_{\mathsf{M\text{-}MAC}}$). We define these games in a similar manner to the games used by Boneh et al. [2001] to derive lower bounds on the security of multicast message authentication. That is, in each game, the adversary sends to the challenger a target message and a subset of corrupted parties that collude to fool one or more receivers; further, the challenger answers the adversary's queries and the game ends with the adversary delivering a forged authentication tag. The attack on unforgeability is classical and the proof is straightforward from the property of the underlying MAC function. The attack on strong nonmalleability $\mathsf{Game}^{\mathsf{SNM}}_{\mathsf{M\text{-}MAC}}$ requires an adversary to be able to modify an M-MAC in such way that verification fails with at least one of the keys but succeeds with another. We call an M-MAC that is resilient to such attacks *strongly nonmalleable*.

Since the unforgeability attack game is easier to derive, we begin with the second property. We prove this by means of a sequence of games following a transition based on the indistinguishability between the key derivation function and some complete random function, a crisper way for providing a security proof [Shoup 2004].

For simplicity of the exposition, we use $\widetilde{m}$ to denote the input to the Tag algorithm and we assume it stands for a tuple of s identical messages, that is, $\mathbb{M} = \{\underbrace{\widetilde{m}, \widetilde{m}, \ldots, \widetilde{m}}_{s-times}\}$.

Definition 1 (*Strong Nonmalleability Attack Game*). *We define the* M-MAC *strong nonmalleability game* $\mathsf{Game}^{\mathsf{SNM}}_{\mathsf{M\text{-}MAC}}$ *as the following three-stage game between challenger* $\mathcal{C}$ *and adversary Adv:*

(*1*) *Setup stage:*
  (*a*) *Challenger* $\mathcal{C}$ *runs the key generation algorithm* $\mathsf{Gen}(1^\ell, s)$ *to get a key set* $\mathbb{K} = \{\mathbb{k}_1, \ldots, \mathbb{k}_s\}$ *of* ℓ*-bit keys (*ℓ *is the security parameter of the scheme).*
  (*b*) *The adversary Adv makes its corruption query by requesting* $\mathcal{C}$ *a subset of the keyset* $\mathbb{K}^\spadesuit = \{\mathbb{k}_{j_1}, \ldots, \mathbb{k}_{j_t}\}$*, where* $j_i \in [1..s]$ *and* $t < s - 1$*. Note that in this case the adversary is always missing at least two of the keys.*
  (*c*) *The adversary Adv fixes its target message* $\widetilde{m}^\blacklozenge$ *and two target key indexes* α, β *such that* $\mathbb{k}_\alpha, \mathbb{k}_\beta \notin \mathbb{K}^\spadesuit$ *(i.e., these two keys are not corrupted).*

(2) *Query stage:*
  (*a*) *Adversary Adv is allowed to make queries to the MAC generation oracle* $\mathcal{O}^{\mathsf{Tag}}(\mathbb{K}, \widetilde{m})$ *for any message tuple* $\widetilde{m}$ *to obtain the corresponding tag* $\tau = \{x_1, x_2, \ldots, x_s\}$*, where each* x_i *is the solution of the following linear system in GF*(2^b)*:*

$$\begin{cases} \mathsf{KD}_1(\mathbb{k}_1, \widetilde{m}) \cdot x_1 + \cdots + \mathsf{KD}_s(\mathbb{k}_1, \widetilde{m}) \cdot x_s \equiv \mathsf{MAC}_{\mathbb{k}_1}\widetilde{m}) \\ \mathsf{KD}_1(\mathbb{k}_2, \widetilde{m}) \cdot x_1 + \cdots + \mathsf{KD}_s(\mathbb{k}_2, \widetilde{m}) \cdot x_s \equiv \mathsf{MAC}_{\mathbb{k}_2}(\widetilde{m}) \\ \cdots \\ \mathsf{KD}_1(\mathbb{k}_s, \widetilde{m}) \cdot x_1 + \cdots + \mathsf{KD}_s(\mathbb{k}_s, \widetilde{m}) \cdot x_s \equiv \mathsf{MAC}_{\mathbb{k}_s}(\widetilde{m}). \end{cases}$$

  (*b*) *Adversary Adv is allowed to make queries to the MAC verification oracle* $\mathcal{O}^{\mathsf{Ver}}(i, \widetilde{m}, \tau)$ *with any key index* i*, tag* $\tau = \{x_1, x_2, \ldots, x_s\}$*, and message* $\widetilde{m}$*. The oracle* $\mathcal{O}^{\mathsf{Ver}}(i, \widetilde{m}, \tau)$ *proceeds in the natural way by computing* $\tau' \equiv \mathsf{KD}_1(\mathbb{k}_i, \widetilde{m}) \cdot x_1 + \mathsf{KD}_2(\mathbb{k}_i, \widetilde{m}) \cdot x_2 + \cdots + \mathsf{KD}_s(\mathbb{k}_i, \widetilde{m}) \cdot x_s$ *and checking that* $\tau' = \mathsf{MAC}_{\mathbb{k}_i}(\widetilde{m})$*.*

(*3*) *Output stage:*
  (*a*) *Eventually, the adversary outputs the pair* $(\widetilde{m}^\blacklozenge, \tau^\clubsuit)$ *where* $\tau^\clubsuit$ *is a tag maleated by the adversary.*
  (*b*) *The game output is* 1 *if the following two conditions hold:* Ver *outputs 1 on* $(\mathbb{k}_\alpha, \widetilde{m}^\blacklozenge, \tau^\clubsuit)$ *and* 0 *on* $(\mathbb{k}_\beta, \widetilde{m}^\blacklozenge, \tau^\clubsuit)$*. Otherwise, the game output is* 0*. Note that in this case, the adversary is allowed to query the MAC generation oracle* $\mathcal{O}^{\mathsf{Tag}}(\mathbb{K}, \widetilde{m})$ *with* $\widetilde{m}^\blacklozenge$ *to get the correct tag* $\tau^\blacklozenge$*.*

Definition 2 (*Strong Nonmalleability*). *We say that a mixed message authentication code* M-MAC *is strongly nonmalleable if*

$$\Pr\left[\mathsf{Game}^{\mathsf{SNM}}_{\mathsf{M\text{-}MAC}}(1^\ell, s) = 1\right] < \mathrm{negl}(\ell).$$

THEOREM 5.1. *The linearly mixed message authentication code* LM-MAC *is strongly nonmalleable.*

PROOF. We construct the following sequence of games:

—**Game** $\mathbf{G}_0$. Let this be the original attack game $\mathsf{Game}^{\mathsf{SNM}}_{\mathsf{M\text{-}MAC}}$.

—**Game $\mathbf{G}_1$**. Let this be identical to game $\mathbf{G}_0$ except for the following: in the query stage, whenever solving the linear system of equations, replace $\mathsf{KD}_1(\mathbb{k}_\alpha, \widetilde{m}), \mathsf{KD}_1(\mathbb{k}_\alpha, \widetilde{m}), \ldots, \mathsf{KD}_s(\mathbb{k}_\alpha, \widetilde{m})$ with some pure random values $\rho_1, \rho_2, \ldots, \rho_s$ (for each $\widetilde{m}$ these are stored on a tape).

—**Game $\mathbf{G}_2$**. Let this be identical to game $\mathbf{G}_1$ except that when verifying the tuple $(\alpha, \widetilde{m}^{\blacklozenge}, \tau^{\clubsuit})$, instead of running $\mathcal{O}^{\mathsf{Ver}}(i, \widetilde{m}, \tau)$ in the usual way, the challenger first checks if the adversary ever asked for the correct tag $\tau^{\blacklozenge}$, retrieves it from the tape (if not then run $\mathcal{O}^{\mathsf{Tag}}(\mathbb{K}, \widetilde{m})$ to get it), and checks if $\rho_1 \cdot (x_1^{\clubsuit} - x_1^{\blacklozenge}) + \rho_2 \cdot (x_2^{\clubsuit} - x_2^{\blacklozenge}) + \cdots + \rho_s \cdot (x_s^{\clubsuit} - x_s^{\blacklozenge}) \equiv 0$.

Assuming that the probability in distinguishing between the key derivation function and some pure random function is negligible $\epsilon(\ell)$, we have $\Pr[Adv \text{ wins } \mathbf{G}_0 - Adv \text{ wins } \mathbf{G}_1] < \epsilon(\ell)$.

If $\rho_1 \cdot (x_1^{\clubsuit} - x_1^{\blacklozenge}) + \cdots + \rho_s \cdot (x_s^{\clubsuit} - x_s^{\blacklozenge}) \equiv 0$, then it also holds that $\rho_1 \cdot x_1^{\clubsuit} + \cdots + \rho_s x_s^{\clubsuit} \equiv \mathsf{MAC}_{\mathbb{k}_\alpha}(\widetilde{m})$; thus, whenever the condition of game $\mathbf{G}_2$ is verified, the one for game $\mathbf{G}_1$ is verified as well. It follows that the probability that game $\mathbf{G}_2$ outputs 1 is at least that of game $\mathbf{G}_1$ and we have $\Pr[Adv \text{ wins } \mathbf{G}_1] \leq \Pr[Adv \text{ wins } \mathbf{G}_2]$. However, note that $\exists i \in [1..s]$ such that $x_i^{\clubsuit} \neq x_i^{\blacklozenge}$, since otherwise $\tau^{\clubsuit} = \tau^{\blacklozenge}$ and in this case $x_i^{\clubsuit}$ will also pass verification for key $\mathbb{k}_\beta$, making the adversary loose the game. Therefore, $\exists i \in [1..s]$ such that $x_i^{\clubsuit} - x_i^{\blacklozenge} \neq 0$, and since this is multiplied by some pure random ρ_i value, we have

$$\Pr[Adv \text{ wins } \mathbf{G}_2] = \Pr\left[\sum_{j=1..s} \rho_j \cdot (x_j^{\clubsuit} - x_j^{\blacklozenge}) \equiv 0\right] = \frac{1}{2^b}$$

$$\Rightarrow \Pr[Adv \text{ wins } \mathbf{G}_0] \leq \epsilon(\ell) + \frac{1}{2^b} = \epsilon(\ell) + \frac{1}{2^{p(\ell)}} \leq \text{negl}(\ell).$$

We considered b to be a polynomial $p(\ell)$ in the security level ℓ.

***Definition* 3** (*Unforgeability Attack Game*). *We define the* M-MAC *unforgeability game* $\mathsf{Game}^{\mathsf{UF}}_{\mathsf{M\text{-}MAC}}$ *as the following five-stage game between challenger $\mathcal{C}$ and adversary Adv:*

(*1*) *Setup stage: is identical to that of the strong nonmalleability game* $\mathsf{Game}^{\mathsf{SNM}}_{\mathsf{M\text{-}MAC}}$ *with the following modification. We request that $t < s$ (i.e., the adversary can get all except one of the keys) and the adversary Adv fixes its target message $\widetilde{m}^{\blacklozenge}$ and a single key index α such that $\mathbb{k}_\alpha \notin \mathbb{K}^{\spadesuit}$.*

(*2*) *Query stage: is identical to that of the strong nonmalleability game* $\mathsf{Game}^{\mathsf{SNM}}_{\mathsf{M\text{-}MAC}}$.

(*3*) *Output stage: the adversary outputs the pair $(\widetilde{m}^{\blacklozenge}, \tau^{\clubsuit})$, where $\tau^{\clubsuit}$ and the game output 1 if the following two conditions hold:* Ver *outputs 1 on $(\mathbb{k}_\alpha, \widetilde{m}^{\blacklozenge}, \tau^{\clubsuit})$ and the adversary never queried $\widetilde{m}^{\blacklozenge}$ to $\mathcal{O}^{\mathsf{Tag}}(\mathbb{K}, \widetilde{m})$. Otherwise, the game outputs 0.*

***Definition* 4** (*Unforgeability*). *We say that a mixed message authentication code* M-MAC *is unforgeable if*

$$\Pr\left[\mathsf{Game}^{\mathsf{UF}}_{\mathsf{M\text{-}MAC}}(1^\ell, s) = 1\right] < \text{negl}(\ell).$$

THEOREM 5.2. *The linearly mixed message authentication code* LM-MAC *is unforgeable if the underlying* MAC *is unforgeable.*

PROOF. Unforgeability is straightforward to relate to the unforgeability of a single MAC. In the setup stage, challenger $\mathcal{C}$ after receiving the index α from the adversary *Adv* solves all future calls to the generation oracle $\mathcal{O}^{\mathsf{Tag}}(\mathbb{K}, \widetilde{m})$ and verification oracle

$\mathcal{O}^{\mathsf{Ver}}(i, \tau, \widetilde{m})$ by using the black box for the MAC. When the adversary outputs $(i, \widetilde{m}^{\blacklozenge}, \tau^{\blacklozenge})$, the challenger yields $\mathsf{MAC}_{\mathbb{k}_\alpha}(\widetilde{m}) = \mathsf{KD}_1(\mathbb{k}_\alpha, \widetilde{m}) \cdot x_1^{\clubsuit} + \mathsf{KD}_2(\mathbb{k}_\alpha, \widetilde{m}) \cdot x_2^{\clubsuit} + \cdots + \mathsf{KD}_s(\mathbb{k}_\alpha, \widetilde{m}) \cdot x_s^{\clubsuit}$, which is a valid tag if and only if the adversary $\mathcal{A}dv$ is successful. Note that similar to the previous games, since the adversary is not in possession of $\mathbb{k}_\alpha$, all values $\mathsf{KD}_i(\mathbb{k}_\alpha, \widetilde{m})$, $i = 1..s$ are simulated by random coins.

ACKNOWLEDGMENTS

We thank Prof. Zonghua Gu for bringing the works of Szilagyi and Koopman [2010] and Szilagyi [2012] to our attention. We are grateful to the anonymous referees who provided useful comments on earlier versions of this manuscript.

REFERENCES

H. Bar-El. 2009. Intra-vehicle information security framework. In *Proceedings of 9th Embedded Security in Cars Conference (ESCAR'09)*.

S. Bittl. 2014. Attack potential and efficient security enhancement of automotive bus networks using short MACs with rapid key change. In *Communication Technologies for Vehicles*. Springer, 113–125.

D. Boneh, G. Durfee, and M. Franklin. 2001. Lower bounds for multicast message authentication. In *Advances in Cryptology (EUROCRYPT'01)*. Springer, 437–452.

A. Bruni, M. Sojka, F. Nielson, and H. R. Nielson. 2014. Formal security analysis of the MaCAN protocol. In *Integrated Formal Methods*. Springer, 241–255.

R. Canetti, J. Garay, G. Itkis, D. Micciancio, M. Naor, and B. Pinkas. 1999. Multicast security: A taxonomy and some efficient constructions. In *Proceedings of 18th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM'99)*, Vol. 2. IEEE, 708–716.

G. Cena and A. Valenzano. 1999. Overclocking of controller area networks. *Electronics Letters* 35, 22 (Oct. 1999), 1923–1925.

H. Chan, A. Perrig, and D. Song. 2003. Random key predistribution schemes for sensor networks. In *Proceedings of the IEEE Symposium on Security and Privacy*. IEEE, 197–213.

L. S. Charlap, H. D. Rees, and D. P. Robbins. 1990. The asymptotic probability that a random biased matrix is invertible. *Discrete Mathematics* 82, 2 (1990), 153–163.

S. Checkoway, D. McCoy, B. Kantor, D. Anderson, H. Shacham, S. Savage, K. Koscher, A. Czeskis, Fr. Roesner, and T. Kohno. 2011. Comprehensive experimental analyses of automotive attack surfaces. In *USENIX Security*.

A. Fiat and M. Naor. 1994. Broadcast encryption. In *Advances in Cryptology (CRYPTO'93)*. Springer, 480–491.

B. Groza, P.-S. Murvay, A.Van Herrewege, and I. Verbauwhede. 2012. LiBrA-CAN: A lightweight broadcast authentication protocol for controller area networks. In *Proceedings of the 11th International Conference on Cryptology and Network Security (CANS'12)*. Springer-Verlag, LNCS.

B. Groza and S. Murvay. 2013. Efficient protocols for secure broadcast in controller area networks. *IEEE Transactions on Industrial Informatics* 9, 4 (2013), 2034–2042.

O. Hartkopp, C. Reuber, and R. Schilling. 2012. Macan-message authenticated can. In *Proceedings of the 10th International Conference on Embedded Security in Cars (ESCAR'12)*.

T. Hoppe and J. Dittman. 2007. Sniffing/replay attacks on CAN Buses: A simulated attack on the electric window lift classified using an adapted CERT taxonomy. In *Proceedings of the 2nd Workshop on Embedded Systems Security (WESS'07)*.

T. Hoppe, S. Kiltz, and J. Dittmann. 2008. Security threats to automotive CAN networks–practical examples and selected short-term countermeasures. In *Proceedings of 27th International Conference on Computer Safety, Reliability, and Security.* 235–248.

International Organization for Standardization 2003. *ISO 11898-1. Road Vehicles - Controller Area Network (CAN) - Part 1: Controller Area Network Data Link Layer and Medium Access Control*. International Organization for Standardization.

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

R. Kurachi, Y. Matsubara, H. Takada, N. Adachi, Y. Miyashita, and S. Horihata. 2014. CaCAN - Centralized authentication system in CAN (controller area network). In *Proceedings of the 14th International Conference on Embedded Security in Cars (ESCAR'14)*.

J. Leohold. 2004. Communication requirements for automotive systems. In *Proceedings of the Keynote Speech 5th IEEE International Workshop on Factory Communication Systems*. Vienna University of Technology.

C.-W. Lin, Q. Zhu, and A. Sangiovanni-Vincentelli. 2015. Security-aware modeling and efficient mapping for CAN-based real-time distributed automotive systems. *IEEE Embedded Systems Letters* 7, 1 (2015), 11–14.

M. Naor and B. Pinkas. 1998. Threshold traitor tracing. In *Advances in Cryptology (CRYPTO'98)*. Springer, 502–517.

A. Perrig, R. Canetti, D. Song, and J. D. Tygar. 2001. SPINS: Security protocols for sensor networks. In *Proceedings of the 7th Annual ACM International Conference on Mobile Computing and Networks (MobiCom'01)*. 189–199.

A. Perrig, R. Canetti, J. D. Tygar, and D. X. Song. 2000. Efficient authentication and signing of multicast streams over lossy channels. In *Proceedings of the IEEE Symposium on Security and Privacy*. 56–73.

Robert BOSCH GmbH 1991. *CAN Specification Version 2.0.* Robert BOSCH GmbH.

Robert BOSCH GmbH 2012. *CAN with Flexible Data-Rate Version 1.0*. Robert BOSCH GmbH.

T. Roeder, R. Pass, and F. B. Schneider. 2012. Multi-verifier signatures. *Journal of Cryptology* 25, 2 (2012), 310–348.

V. Shoup. 2004. Sequences of games: A tool for taming complexity in security proofs. *IACR Cryptology ePrint Archive* 2004 (2004), 332.

I. Studnia, V. Nicomette, E. Alata, Y. Deswarte, M. Kaâniche, and Y. Laarouchi. 2013. Survey on security threats and protection mechanisms in embedded automotive networks. In *Proceedings of the 43rd Annual IEEE/IFIP Conference on Dependable Systems and Networks Workshop (DSN-W'13)*. IEEE, 1–12.

C. Szilagyi and P. Koopman. 2009. Flexible multicast authentication for time-triggered embedded control network applications. In *Proceedings of the IEEE/IFIP International Conference on Dependable Systems & Networks*. IEEE, 165–174.

C. Szilagyi and P. Koopman. 2010. Low cost multicast authentication via validity voting in time-triggered embedded control networks. In *Proceedings of the 5th Workshop on Embedded Systems Security*. ACM, 10.

C. J. Szilagyi. 2012. *Low Cost Multicast Network Authentication for Embedded Control Systems*. Ph.D. Dissertation. PhD Thesis, Carnegie Mellon University.

A. Van Herrewege, D. Singelee, and I. Verbauwhede. 2011. CANAuth-A simple, backward compatible broadcast authentication protocol for CAN bus. In *Proceedings of the 9th Embedded Security in Cars Conference*.

Q. Wang and S. Sawhney. 2014. VeCure: A practical security framework to protect the CAN bus of vehicles. In *Proceedings of the International Conference on the Internet of Things (IOT'14)*. IEEE, 13–18.

M. Wolf, A. Weimerskirch, and C. Paar. 2006. Secure in-vehicle communication. *Embedded Security in Cars*. Springer, 95–109.

S. Woo, H. J. Jo, and D. H. Lee. 2015. A practical wireless attack on the connected car and security protocol for in-vehicle CAN. *IEEE Transactions on Intelligent Transportation Systems* 16, 2 (2015), 993–1006.

T. Ziermann, S. Wildermann, and J. Teich. 2009. CAN+: A new backward-compatible Controller Area Network (CAN) protocol with up to 16x higher data rates. In *Proceedings of Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 1088–1093.

Received October 2014; revised February 2016; accepted November 2016