Efficient Protocols for Secure Broadcast in Controller Area Networks

Bogdan Groza, *Member, IEEE*, and Stefan Murvay, *Student Member, IEEE*

***Abstract*—Controller Area Network is a bus commonly used by controllers inside vehicles and in various industrial control applications. In the past controllers were assumed to operate in secure perimeters, but today these environments are well connected to the outside world and recent incidents showed them extremely vulnerable to cyber-attacks. To withstand such threats, one can implement security in the application layer of CAN. Here we design, refine and implement a broadcast authentication protocol based on the well known paradigm of using key-chains and time synchronization, a commonly used mechanism in wireless sensor networks, which allows us to take advantage from the use of symmetric primitives without the need of secret shared keys during broadcast. But, as process control is a time critical operation we make several refinements in order to improve on the authentication delay. For this we study several trade-offs to alleviate shortcomings on computational speed, memory and bandwidth up to the point of using reduced versions of hash functions that can assure ad hoc security. To prove the efficiency of the protocol we provide experimental results on two representative microcontrollers from the market: a Freescale S12X and an Infineon TriCore, both devices were specifically chosen as they are located somewhat on the extremes of computational power.**

***Index Terms*—Authentication, broadcast, CAN, S12X, TriCore.**

I. Motivation and Related Work

CONTROLLER Area Network or simply CAN is a communication bus frequently used in vehicular systems and also common in general purpose automations. Initially developed by BOSCH, the current specifications for CAN are found in the newer standard ISO-11898. Numerous further improvements appeared in the literature: dual channel architectures [9], flexible time-triggered communication [1], star topologies [4], [5], dynamic identifiers to improve timing requirements [8], domain specific adaptations in the aeronautical sector [26], etc.

In the last decade, industrial control and automation systems become open to malicious adversaries and a significant part of the security community focused on alleviating potential threats in such environments [13], [15]. Also, recent incidents of international level, such as the Stuxnet worm, have shown that embedded devices are not as isolated as once thought and can become vulnerable targets [14]. As for in-vehicle security, recent research showed current automobiles to be unexpectedly vulnerable to external adversaries [11], [18] and it is likely that many other environments in which CAN operates are not completely isolated from the outside world. Two comprehensive books for security in automotives and cryptography based solutions in particular are [20] and [17], both contain relevant chapters on intra-vehicle security. Also several research papers published in the last years address these issues [3]. Still, to best of our knowledge there is no implementation available to assure authenticity in CAN networks, while the secure perimeters in which CAN used to operate are becoming increasingly open to the outside world.

To withstand such threats, security can be implemented at the application level of CAN. For this we study several trade-offs to achieve a potentially optimal solution. Digital signatures provide an elegant method for signing broadcast data, but they are not the solution in our context because of both the computational and communication overhead. Elliptic curves can significantly reduce the size of the signatures, but still the computational overhead is too much to assure small authentication delays. While the computational overhead can be alleviated by dedicated circuits, such as ASICs and FPGAs, this will increase the cost of components, an issue that is largely avoided by manufacturers. One alternative to digital signatures such as RSA, or ECDSA is the use of one-time signatures which were initially proposed by Merkle [24]. Although they are frequently mentioned in the literature, they are quite unused in practice mostly because of their one-time nature and less favourable re-initialization. In contrast, symmetric primitives were efficiently employed for authentication in constrained environments such as sensor networks [21], [22], [28]. Due to the broadcast nature of CAN, protocols similar to the well known TESLA protocol [27], [29] can be used in this context as well. Indeed, some of the constraints are similar. For example, computational power is also low and, although high speed microcontrollers are also available on the market, low speed microcontrollers are preferred to reduce costs. While TESLA like protocols introduce delays that could be unsuitable for all real-time CAN based applications, there is a broad area of applications where they could be tolerated in exchange for security. In particular, delays in the order of milliseconds, or even below as proved to be achievable by our proof-of-concept implementation, are suitable for a broad area of real-time control tasks. Many examples of network control scenarios that can accommodate such delays can be found in the literature [23].

There is an extensive bibliography related to the TESLA protocol, its history can be traced back to Lamport's authentication scheme [19]. The work of Bergadano *et al.* [7] proposes several variants of one-way chain based protocols, with or without

Manuscript received September 20, 2011; revised January 22, 2012; accepted November 29, 2012. Date of publication January 11, 2013; date of current version October 14, 2013. This work was supported in part by CNCSIS-UEFISCDI project number PNII-IDEI 940/2008 and by the strategic grant POSDRU 107/1.5/S/77265 and POSDRU/21/1.5/G/13798. Paper no. TII-11-543.

The authors are with the Faculty of Automatics and Computers, Politehnica University of Timisoara, Romania (e-mail: bogdan.groza@aut.upt.ro; stefan.murvay@gmail.com).

Digital Object Identifier 10.1109/TII.2013.2239301

time synchronization. In particular, several trade-offs for sensor networks were studied by Liu and Ning in [21], [22] and several variants of the protocols are presented by Perrig *et al.* as well in [27], [29]. Also several papers addressed hybrid versions in which asymmetric primitives are mixed with key-chains in order to obtain trade-offs [2], [6], [10]. However, these variants have a bigger communication or computational overhead and do not appear to be appropriate for our application setting.

Contribution. The main argument for choosing a TESLA like protocol in our research here is that this is the best solution to perform broadcast authentication without secret shared keys or public key primitives. Also, to best of our knowledge, there is no result so far that points out clear technical limits on using TESLA like protocols on CAN networks. Thus, we provide clear experimental results on two microcontrollers located somewhat on the extremes of computational power, memory and bus speed: an S12 equipped with an XGATE coprocessor and an Infineon TriCore. In previous work [16] we used a classical multi-layer key-chain in order to authenticate broadcast in CAN. Compared to our previous approach, we improve here the performance by almost 2 orders of magnitude. This is achieved first by using two schemes that have ad hoc security and use reduced versions of hash functions. Second, we used for the experimental setup a stronger microcontroller from Infineon capable of high-speed CAN that works up to 1 Mbps, while in previous work we used only the fault tolerant version of CAN limited to 125 kbps. While our previous result allowed an authentication delay of around 10 ms, we make the authentication delay here to drop in the order of 300 μs. The improvements provided here are relevant as the authentication delay is critical for control scenarios. While in sensor networks the disclosure interval is usually in the order of tens or hundreds of milliseconds here we bring this delay lower by 2 to 3 orders of magnitude. Depicting an optimal protocol configuration in this context is not straight forward and we study several trade-offs in what follows.

II. Environment and Protocol Description

CAN is a two wire broadcast bus with arbitration based on the frame identifier ID which has 29 bits in extended frames and 11 bits in standard frames. The structure of the CAN frame consists in the arbitration field (referred in what follows as the identifier ID), 6 bit control field, 0–64 bits of data, 15 bit CRC and a 2 bit acknowledgement. Arbitration based on message priority is a relevant feature of the bus, but it is transparent to our protocol from an upper view. In our implementation, we just make sure that keys and other security elements have the highest priority. There is also active interest toward using star topologies in CAN networks [4], [5] and this would have certain advantages from a security perspective. The identifiers are generally fixed and are not changed during runtime, however assigning dynamic IDs was suggested in the past [8] in order to improve on timing requirements and can be considered in further optimizations for particular scenarios.

Indeed, CAN is a message oriented bus while TESLA appears to be source oriented, i.e., it assures that a message originates from a particular sender. We emphasize that there are many practical scenarios in which the source of a particular

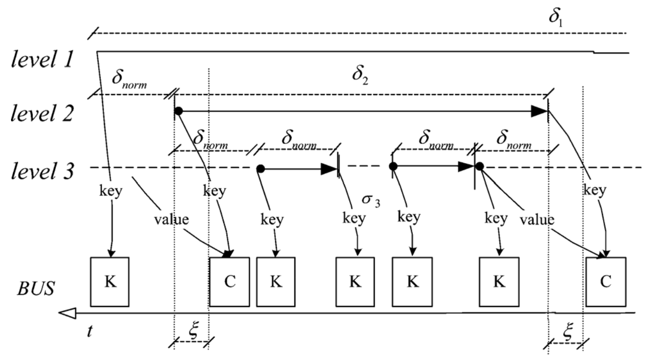

Fig. 1. Broadcast sequence with normalized time δ_{norm}.

CAN message does matter and in practice identifiers are frequently uniquely associated to a particular node. Thus the message oriented nature of CAN should not be interpreted in a strict sense, where the source of the message is irrelevant. Even for the case of an ID oriented authentication (where authentic messages with the same ID can originate from different nodes) a TESLA like protocol will prove to be more suitable for adding new nodes on the bus since they can authenticate messages via the broadcast scheme without needing to share the secret key for a particular ID.

From an upper view, our design paradigm is the following. Memory, computational speed, bandwidth, initialization time and the synchronization error give bounds on the structure of the chains that we can use. This further bounds the authentication delay, i.e., the delay at which authentication keys arrive on the bus, which is crucial to us as messages cannot be authenticated faster than the disclosure delays. All protocol variants use multiple levels of one-way key chains with the structure suggested in Fig. 1. The relevant notations with respect to the chain structure are: ℓ the number of chain levels, $\sigma_i,\ i = 1\ldots\ell$ the length of the chain on level i, $\delta_i,\ i = 1\ldots\ell$ the disclosure delay on level i, ξ the safety margin for releasing authentication packets and δ_{norm} the *normalized disclosure delay* (these are further detailed in the next section). Informally, bullets depict keys from the key chains and the horizontal black arrows denote that they are derived from a previous key. Keys are generated and consumed in a reverse order, thus the time arrow on the bus points in opposite direction to the arrows that generate the keys. Packets arriving on the bus are depicted as well, the dotted lines from a chain element to the packet denote that the element was used as a key. For the re-initialization packets, one element of the key chain was also used as a message. Packets containing keys are marked by K and commitments, i.e., MAC codes that authenticate forthcoming key chains, are marked by C.

Before the broadcast protocol can run, we need an initialization protocol. Its role is to allow each unit to commit or retrieve its initialization values and to achieve time synchronization with the sender. This part of the protocol can also rely on more expensive algebraic operations required by public-key cryptography. Although public key certificates are larger and will require more than one frame (which can carry at most 64 bits) in general it should not be a problem to transport them over CAN if this does not happen too often and just in the initialization stage. If public keys are not a feasible alternative, then initialization keys can be hardcoded in an off-line manner. Time synchronization is done with respect to a central node, which will play the role of a communication master. As usual, synchronization between two nodes is loose and it requires a handshake and counting the round trip time until it is below a tolerance margin. To determine the effect of the bus speed on the synchronization procedure, we measured the round-trip time for a short message exchange. The slave node that wants to start communication with the master first sends a nonce to the master. Upon receiving this request, the master sends it's response containing the MAC computed over the nonce. The measured round-trip time is the time span between the moment in which the slave sends the nonce and the moment the reception of the response from the master is over. Our measurements lead to synchronization errors in the order of 3–5 ms for S12X and 200–300 μs for TriCore.

A. Generic Description of the Protocol

For the sender side, we first define the structure of the key chain with respect to each level and then we define the precise timings for the disclosure of each key. We make use of a *timing template* which is used to compute the timings for each level (based on chain lengths and disclosure delays) and a *function template* which is used to generate the keys on each level (based on a one-way function). Different to previous work, we use the *function template* to allow the generation of chains from different levels, with different functions, that will provide good speed-ups in the following variants.

Definition 1: We define the *timing template* as an ℓ-tuple of positive integer pairs denoting the chain length and disclosure delay for each particular level, i.e., $\mathbf{T}_\ell = \{(\sigma_1, \delta_1), (\sigma_2, \delta_2), \ldots, (\sigma_\ell, \delta_\ell)\}$.

Definition 2: We define the *function template* as an ℓ-tuple of functions that are used to generate the keys on each level, i.e., $\mathbf{F}_\ell = \{\mathsf{F}_1, \ldots, \mathsf{F}_\ell\}$.

Definition 3: We define the *indexed key collection* $\mathbb{K}_{\mathbf{T},\mathbf{F}}$ as a tuple of *time-indexed keys* K_τ, i.e., keys entailed by a vector τ with ℓ elements that defines the exact disclosure time for the key. Given *timing template* $\mathbf{T}_\ell$, *function template* $\mathbf{F}_\ell$ a *time-indexed key* is generated as: $\mathsf{K}_{\tau_{left}|\tau_i|\bar{0}} = \mathsf{F}_i(\mathsf{K}_{\tau_{left}|\tau_i+1|\bar{0}}), \forall i \in [1, \ell], \tau_i \in [0, \sigma_i - 1]$.

Here $\mathsf{K}_{\tau_{left}|\sigma_i|\bar{0}}$ is the initialization key for the particular key-chain, computed via a key-derivation process from some random fresh material generated at each initialization as $\mathsf{K}_{\tau_{left}|\sigma_i|\bar{0}} = \mathsf{KD}(\mathsf{K}_{\mathrm{rnd}}, \tau_{left})$, $\mathsf{K}_{\mathrm{rnd}}$ is some fixed random value and KD is a key derivation function. Here $\tau left$ is a placeholder for any left part of the index vector τ and the right part, denoted by $\bar{0}$, is always zero.

The previous definition allows the generation of chains on multiple levels with the specified length as suggested in Fig. 1. Now we can establish the exact disclosure time for each key. For this we let t_{start} denote the time at which the broadcast was started on the sender side and assuming there are no clock drifts for the sender the exact release time of the keys follows.

Definition 4: Let $\mathcal{DT}(\mathsf{K}_\tau, \mathbf{T}_\ell)$ denote the disclosure time of the *indexed key* K_τ based on *timing template* $\mathbf{T}_\ell$. Given a broadcast started at $tstart$ the disclosure time of K_τ is: $\mathcal{DT}(\mathsf{K}_\tau, \mathbf{T}_\ell) = t_{start} + \sum_{i=1..\ell}(\delta_i \cdot \tau_i)$.

We consider the case of a receiver $\mathcal{R}$ and sender $\mathcal{S}$ with synchronization error $\epsilon_{\mathcal{S},\mathcal{R}}$. Now we define the security condition that must be met by all packets that contain authentication information, i.e., MAC codes, produced with an indexed key K_τ.

Definition 5: Given synchronization error $\epsilon_{\mathcal{S},\mathcal{R}}$ and $t_\mathcal{S}$ the time value reported by $\mathcal{S}$ on a synchronization performed at t_{sync} with $\mathcal{R}$, the minimum and maximum time on the $\mathcal{S}$'s side, *estimated* by $\mathcal{R}$ having local clock pointing at $t_\mathcal{R}$ are: $\mathcal{ET}_{\min}(t_\mathcal{R}) = t_\mathcal{R} - t_{sync} + t_\mathcal{S}$, $\mathcal{ET}_{\max}(t_\mathcal{R}) = t_\mathcal{R} - t_{sync} + t_\mathcal{S} + \epsilon_{\mathcal{S},\mathcal{R}}$.

Definition 6 (Security Condition): Given *timing template* $\mathbf{T}_\ell$, an authentication packet computed with K_τ received at time $t_\mathcal{R}$ must be discarded unless: $\mathcal{ET}_{\max}(t_\mathcal{R}) \leq \mathcal{DT}(\mathsf{K}_\tau, \mathbf{T}_\ell)$.

This condition ensures that an authentication packet is not accepted after the authentication key was already disclosed. The generic description of the protocol now follows. We underline that this description does not include particular optimizations that are presented in the section dedicated to practical variants. It works only as a high level description for the forthcoming protocols.

Definition 7: Given *indexed key collection* $\mathbb{K}_{\mathbf{T},\mathbf{F}}$ and two roles called sender and receiver denoted by $\mathcal{S}$ and $\mathcal{R}$ each with synchronization error $\epsilon_{\mathcal{S},\mathcal{R}}$, protocol $\mathsf{Broadcast}_{\mathcal{S},\mathcal{R}}[\mathbb{K}_{\mathbf{T},\mathbf{F}}]$ is defined by the following two rules for the two roles: i) $\mathcal{S}$ sends K_τ at $\mathcal{DT}(K_\tau)$ and $MAC(K_\tau, M)$ in any empty time-slot with the condition that $MAC(K_\tau, M)$ is released no latter than $\mathcal{DT}(K_\tau) + \xi$. Message M can be released at any time, ii) $\mathcal{R}$ discards all $MAC(K_\tau, M)$ received at $t_\mathcal{R}$ for which $\mathcal{ET}_{\max}(t_\mathcal{R}) \leq \mathcal{DT}(\mathsf{K}_\tau, \mathbf{T}_\ell)$ does not hold and deems authentic all other messages for which $MAC(K_\tau, M)$ can be verified with an authentic key. A key $K_{\tau left|\tau_i|\bar{0}}$ is authentic if and only if $K_{\tau left|\tau_i|\bar{0}} = \mathcal{F}(K_{\tau left|\tau_i+1|\bar{0}})$ and $K_{\tau left|\tau_i+1|\bar{0}}$ is a previously received authentic key (note that $K_{\tau left|0|\bar{0}}$ must be committed via a MAC).

Here ξ denotes a tolerance margin for the time at which a MAC with a particular key can be sent. Indeed, sending MACs too close to the disclosure time may be useless because the receiver may have to discard them if the security condition cannot be met. Thus ξ must be fixed as initial setup parameter for the protocol. In time interval $[\mathcal{DT}(K_\tau), \mathcal{DT}(K_\tau) + \xi]$ the sender can safely disclose any kind of data packet, but not MACs. $\mathsf{Broadcast}_{\mathcal{S},\mathcal{R}}[\mathbb{K}_{\mathbf{T},\mathbf{F}}]$ is a secure broadcast authentication protocol. The security of this family of protocols is well established, formal proofs of security can be found in [7] and [29]. The informal argument is that from $\mathrm{MAC}_\mathcal{K}(M)$ and $\mathsf{F}(\mathcal{K})$ an adversary cannot produce $\mathrm{MAC}_\mathcal{K}(M')$ for any $M' \neq M$ since $\mathcal{K}$ is not known as well as it cannot be found from $\mathsf{F}(\mathcal{K})$. By the time $\mathcal{K}$ is released it is already too late for the adversary to send a MAC and a message as they will not be anymore accepted by the receiver due to the time constraint.

B. Efficiency Parameters

The efficiency of the protocol can be evaluated with respect to memory, CPU and bandwidth. This evaluation has to be done over the entire time horizon of the protocol which can be divided in two parts: initialization time $\mathsf{T}_{\mathsf{init}}$ and runtime $\mathsf{T}_{\mathsf{run}}$. However, bus loads and CPU utilizations, that are going to be defined next, are more relevant only over $\mathsf{T}_{\mathsf{run}}$ as it is natural to expect that during T_{init} the initialization can takeover the entire bus and CPU but only for a very short period of time. We will use the following notations: MEM, CPU, BUS and their capacities are depicted in the number of keys that can be stored, computed or sent. For all of these notations, a subscript indicates whether they refer to the initialization stage or the runtime stage. Thus $\mathsf{CPU}_{\text{init}}$ refers to the amount of work during initialization and $\mathsf{CPU}_{\text{run}}$ during runtime. By $\mathsf{MEM}^{\text{total}}$, $\mathsf{CPU}^{\text{total}}$ and $\mathsf{BUS}^{\text{total}}$ we refer to the total available computational power and bus capacity during the entire run-time of the protocol—we use these measures to define CPU and bus loads during runtime. To indicate whether a resource is needed for computing keys or commitments through MAC codes we use key and com as superscripts.

Definition 8: Given *key collection* $\mathbb{K}_{\mathbf{T},\mathbf{F}}$ we let $\| \mathbb{K}_{\mathbf{T},\mathbf{F}}[i] \|$ denote the total number of keys on level i disclosed during protocol lifetime and $\langle \mathbb{K}_{\mathbf{T},\mathbf{F}}[i] \rangle$ the number of key chains from level i.

Obviously we have $\| \mathbb{K}_{\mathbf{T},\mathbf{F}}[i] \| = \sigma_i \cdot \langle \mathbb{K}_{\mathbf{T},\mathbf{F}}[i] \rangle$ since the total number of keys is the number of chains multiplied with the chain length. We will use both notations, although it is easy to derive one from the other, in order to make the following relations more intuitive.

Let $\{(c_0, m_0), (c_1, m_1), \ldots, (c_\ell, m_\ell)\}$ define the CPU and memory requirements for all elements of the *function template* $\mathbf{F}$. For protocol $\mathsf{Broadcast}_{\mathcal{S},\mathcal{R}}[\mathbb{K}_{\mathbf{T},\mathbf{F}}]$ we define the following overheads. The memory requirements consist in the sum of the lengths of the chains, i.e., $\mathsf{MEM}^{\text{key}}_{\text{init}} = \mathsf{MEM}^{\text{key}}_{\text{run}} = \sum_{i=1,\ell} \sigma_i \cdot m_i$. In the case of memory there are no variations during initialization and runtime. More, we do not need additional memory to store commitments on the sender as commitments can be sent as soon as they are computed. The computational time required for keys at runtime and initialization is $\mathsf{CPU}^{\text{key}}_{\text{init}} = \sum_{i=1,\ell} \sigma_i \cdot c_i$ and $\mathsf{CPU}^{\text{key}}_{\text{run}} = \sum_{i=2,\ell} c_i \cdot (\| \mathbb{K}_{\mathbf{T},\mathbf{F}}[i] \| - \sigma_i)$ respectively. In the initialization one chain is computed on each level. At runtime, there are $\langle \mathbb{K}_{\mathbf{T},\mathbf{F}}[i] \rangle$ key-chains on each level, and each of them has to be computed except the first one which was computed during initialization which gives $\mathsf{CPU}^{\text{key}}_{\text{run}} = \sum_{i=2,\ell} c_i \cdot \sigma_i \cdot (\langle \mathbb{K}_{\mathbf{T},\mathbf{F}}[i] \rangle - 1)$. Replacing $\sigma_i \cdot \langle \mathbb{K}_{\mathbf{T},\mathbf{F}}[i] \rangle$ with $\| \mathbb{K}_{\mathbf{T},\mathbf{F}}[i] \|$ we get the claimed number of keys computed at runtime. The computational costs of commitments consist in one commitment on each level during initialization and later one commitment for each chain on each level (except for the first one which was committed during initialization). This gives $\mathsf{CPU}^{\text{com}}_{\text{init}} = \sum_{i=1,\ell} c_i$ and $\mathsf{CPU}^{\text{com}}_{\text{run}} = \sum_{i=2,\ell} c_i \cdot (\langle\langle \mathbb{K}_{\mathbf{T},\mathbf{F}}[i] \rangle\rangle - 1)$. Bus requirements for keys during runtime is given by $\mathsf{BUS}^{\text{key}}_{\text{run}} = \sum_{i=1,\ell} m_i \cdot \| \mathbb{K}_{\mathbf{T},\mathbf{F}}[i] \|$. All keys from all levels are sent on the bus, while there are no keys (just commitments) sent during initialization. Commitments are given by $\mathsf{BUS}^{\text{com}}_{\text{init}} = \sum_{i=1,\ell} m_i$ and $\mathsf{BUS}^{\text{com}}_{\text{run}} = \sum_{i=2,\ell} m_i \cdot (\langle\langle \mathbb{K}_{\mathbf{T},\mathbf{F}}[i] \rangle\rangle - 1)$. One chain on each level is committed in the initialization, and later at runtime all chains are committed except for the first one, same as in the case of computational requirements.

To complete the view on efficiency, we should also define the CPU and bus loads over the entire lifetime of the protocol. Given $\mathsf{RES} \in \{\mathsf{MEM}, \mathsf{CPU}, \mathsf{BUS}\}$, $\mathsf{state} \in \{\mathsf{run}, \mathsf{init}\}$ we define the protocol overheads as: $\mathsf{RES}^{\text{load}}_{\text{state}} = (\mathsf{RES}^{\text{key}}_{\text{state}} + \mathsf{RES}^{\text{com}}_{\text{state}})/\mathsf{RES}^{\text{total}}_{\text{state}}$. One can add to these the overhead induced by the message authentication codes associated to each data packet that is sent over the bus. This is however application dependent, not protocol dependent as in some applications the size of the data packets can be small, and thus adding a MAC to each data packet will greatly increase the overhead while in other applications it may be the reverse, and data packets can be large and the MAC will not significantly increase the overhead.

III. Practical Variants

Now we discuss practical variants of the main scheme. Obtaining a variant that is adequate, possibly optimal, for practical use means to satisfy the constraints of the environment. The *generic calibration criteria* for the scheme parameters is the following. Having fixed T_{run} and δ_{norm} we determine chain structure (lengths and levels) and timings which give $\mathbf{T}_\ell$, $\mathbf{F}_\ell$ and $\mathbb{K}_{\mathbf{T},\mathbf{F}}$. Then, we determine bus, CPU and memory loads for comparison.

A. The Multi Master and Single Master Case

CAN allows each node to be a potential sender. The case of k senders can be easily derived from the previous formalism. We can multiplex the senders by using δ_{next} which we call the *next sender delay*. By this, we can modify the disclosure timings to $\mathcal{DT}_k(K_\tau) = \mathcal{DT}(K_\tau) + k \cdot \delta_{next}$ and the security condition accordingly for the case of k senders.

However, allowing more than one sender will result in a bus that is heavily loaded by keys and commitments. To avoid this, having only one communication master is preferable. In the case when one of the slave nodes needs to broadcast authentic information it can perform a request to the communication master under the assumption that each slave node shares a secret key with the master that can be used for authentication. We can take advantage of the CAN nature as each slave can place its data frame on the bus, along with a message authentication code computed with the shared key. The communication master can verify this MAC and, if correct, it will send a next frame that contains the broadcast authentication information, i.e., the MAC with the current key according to the broadcast protocol. The rest of the protocol is unchanged, we do not give a methodology to compute parameters as this is going to be discussed for the next variants. This approach will not increase the authentication delay if the slave nodes are able to send the message and its MAC during the same disclosure period in which the master can continue to broadcast the authentication MAC.

B. Equidistant Timing Authenticated CAN (ETA-CAN)

For practical reasons, a solution which assures a uniform bus load is preferable. This is mostly because packets carrying data must be delayed until all keys and commitments are sent since they have priority on the bus (otherwise the protocol will succumb and have to be re-initialized).

For this, we use a procedure which we call *equidistant timing* by which all keys are disclosed at moments of time separated by equal delays. This is relevant also because we can use upper layer chains not only to authenticate the commitments of keys from the lower levels but also to authenticate information packets as well. The same *equidistant* release will be used for key commitments. Thus, we will normalize the disclosure time

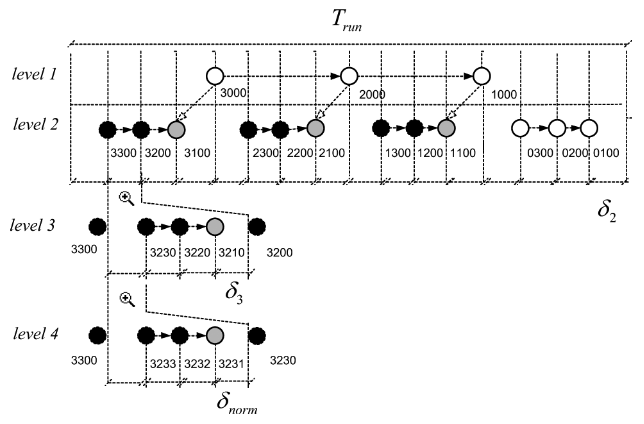

Fig. 2. Chains structure with $BETA$ ($\ell = 4$, $\sigma_{\mathsf{norm}} = 3$, $\delta_{\mathsf{norm}} = \mathsf{T}_{\mathsf{run}}/255$).

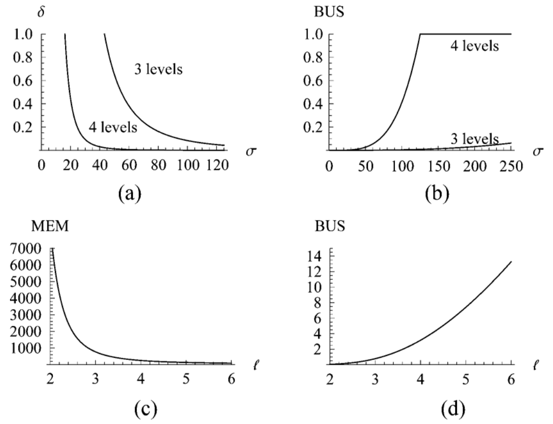

Fig. 3. Various overheads and delay variation with length or levels: (i) disclosure delay, (ii) overhead caused by keys and commitments on the bus, (iii) keys stored in memory and (iv) commitments on the bus (per second).

TABLE I
SOME OVERHEADS AT INITIALIZATION AND RUN-TIME

	BETA-CAN	Ad-BETA-CAN
$\mathsf{MEM}^{\star}_{\mathsf{key}}$	$\ell \cdot \sigma \cdot m$	$\sigma \cdot \sum_{i=1,\ell} m_i$
$\mathsf{CPU}^{\mathsf{init}}_{\mathsf{key}}$	$\ell \cdot \sigma \cdot c$	$\sigma \cdot \sum_{i=1,\ell} c_i$
$\mathsf{CPU}^{\mathsf{init}}_{\mathsf{com}}$	$\ell \cdot c$	$c_0 + \sum_{i=1,\ell-1} c_i$
$\mathsf{BUS}^{\mathsf{init}}_{\mathsf{com}}$	$\ell \cdot m$	$m_0 + \sum_{i=1,\ell-1} m_i$
$\mathsf{CPU}^{\mathsf{run}}_{\mathsf{key}}$	$[(\sigma+1)^{\ell} - \sigma \cdot \ell - 1] \cdot c$	$\sigma \cdot \sum_{i=2,\ell} [(\sigma+1)^{i-1} - 1] \cdot c_i$
$\mathsf{BUS}^{\mathsf{run}}_{\mathsf{key}}$	$[(\sigma+1)^{\ell} - 1] \cdot m$	$\sigma \cdot \sum_{i=1,\ell} (\sigma+1)^{i-1} \cdot m_i$
$\mathsf{CPU}^{\mathsf{run}}_{\mathsf{com}}$	$[\frac{(\sigma+1)^{\ell}-1}{\sigma} - \ell] \cdot c$	$\sum_{i=2,\ell} [(\sigma+1)^{i-1} - 1] \cdot c_i$
$\mathsf{BUS}^{\mathsf{run}}_{\mathsf{com}}$	$[\frac{(\sigma+1)^{\ell}-1}{\sigma} - \ell] \cdot m$	$\sum_{i=2,\ell} [(\sigma+1)^{i-1} - 1] \cdot m_i$

on the last level and then compute the disclosure delays on the upper levels. These disclosure timings are suggested in Fig. 1.

Definition 9: For the ETA-CAN we define the disclosure delays as: $\delta_\ell = \delta_{\mathsf{norm}} = \mathsf{T}_{\mathsf{run}} / \sum_{i=1,\ell} \parallel \mathbb{K}_{\mathbf{T},\mathbf{F}}[i] \parallel = \mathsf{T}_{\mathsf{run}} / \prod_{i=1,\ell}(\sigma_i + 1) - 1$, $\delta_i = \delta_{\mathsf{norm}} \cdot \prod_{j=i+1,\ell}(\sigma_j + 1), 1 \leq i < \ell$.

It is easy to note that given a fixed amount of memory which must accommodate the chains and a fixed number of levels ℓ, the disclosure delay δ_{norm} and the overheads for CPU and bus have an inverse variation. Thus: the minimal disclosure delay is achieved if chains are of equal size while the minimal computational and communication overhead is achieved if upper level chains are smaller. We now explore the variant with chains of equal sizes on all levels called Balanced Equidistant Timing Authenticated CAN (BETA-CAN). To clarify previous notations Fig. 2 shows an example of chain structure for the case of $\sigma = 3$ (note that the same key-chain size is on all levels). Since the entire run-time of the protocol is $\mathsf{T}_{\mathsf{run}} = \delta_{\mathsf{norm}} \cdot [(\sigma+1)^{\ell^{\mathsf{BETA}}} - 1]$ the number of levels follows as: $\ell^{\mathsf{BETA}} = \lceil \log_{\sigma+1}(\mathsf{T}_{\mathsf{run}}/\delta_{\mathsf{norm}} + 1) \rceil$. The disclosure delay of the last level is δ_{norm} while for the upper levels the delay can be computed as: $\delta_i^{\mathsf{BETA}} = \delta_{\mathsf{norm}} \cdot (\sigma + 1)^{\ell^{\mathsf{BETA}} - i}$. Having these defined the performance indicators for memory, CPU and bus can be derived. These indicators are summarized for all variants in Table I.

Fig. 3 shows the influence of chain length and levels on various parameters. Plots are taken for a time range $\mathsf{T}_{\mathsf{run}} = 24$ hours while the bus speed is approximated to about 6000 packets per second. In the case of variations with chain lengths, plots (i) and (ii) are given for three and four levels of key chains. We note that the delays drop rapidly by increasing the number of levels in plot (i), but in the same manner the overhead increases (ii) (at 100% the bus is locked and communication halted). Plot (iii) shows the variation of memory requirements, which is the same as the initialization time, and plot (iv) of commitments with the number of chain levels. The same drop of memory requirements and increase in commitment packets can be seen. For plots (iii) and (iv) the delay is fixed to 5 ms.

C. Ad Hoc Secure Variants of the Scheme

To increase performance we will use reduced versions of hash functions. The following definition is informal and will serve only as a heuristic for the security of the schemes.

We call function F_k *ad hoc secure* with respect to time interval δ if given $y = \mathsf{F}_k(x)$, with $\mathsf{F}_k(x)$ publicly known, it is infeasible to find an x' in time δ such that $y = \mathsf{F}_k(x')$ (note that x' does not necessary need to be same as x since any x' which has the same image under F_k will suffice to break the protocol). Here k plays the role of an index for function F. In our practical implementation we use for function F_k truncated versions of hash functions in order to reduce the overhead on the bus. However, since the image of F is reduced, for example to 32 bits in the worst case, it can be feasible for an adversary to mount a pre-computed dictionary attack. To avoid this, we compute the α-bit truncated hash chain in the following manner: at each step we compute $\mathsf{K}_{\tau_{left}|\tau_i|\bar{0}} = \lfloor \mathsf{F}_k(kd \| \mathsf{K}_{\tau_{left}|\tau_i+1|\bar{0}}) \rfloor_\alpha$. Here kd stands from some material derived from the key template, i.e., previously released keys, in order to assure sufficient entropy against pre-computed attacks, similar to salting. Note that the same truncation can be done for MAC codes under the restriction that messages and keys are not released later than the security lifetime δ (if the message or key is to be released later, then the appropriate size for the MAC is to be chosen).

Note that the disclosure delays are now needed to establish the exact security level that must be met by functions on each level of the chains. Therefore, these delays determine the *function template* that can be used. For reduced variants of the hash functions, only heuristic arguments can be given, that is, protocol $\mathsf{Broadcast}_{\mathcal{S},\mathcal{R}}[\mathbb{K}_{\mathbf{T},\mathbf{F}}]$ is *ad hoc secure* with respect to the corresponding disclosure timings.

Same as in the previous scheme, for the Balanced levels ETA-CAN (Ad-BETA-CAN) we use chains of equal sizes to minimize the number of layers but we select different functions on each level such that the function is *ad hoc* secure with respect to the disclosure delay on the particular level. Thus, given δ_{norm} we first select the less intensive function that is *ad hoc secure* with respect to δ_{norm}. Then we assume that this function is going to be used on all levels, take the constraints on memory and CPU and successively try the smallest value of ℓ until they are satisfied. Then, we proceed from the ℓ-th chain upward to change the function to one that is *ad hoc secure* with the respective delay. If the constraints are not fulfilled we chose a bigger ℓ and so on. The number of levels and the delays are computed in the same manner as previously while performance indicators are summarized as well in Table I. An explanatory example follows after the introduction of the next scheme.

The Ad hoc secure Greedy last level ETA-CAN (Ad-GETA-CAN) variant uses a greedy strategy in order to minimize memory overheads. Given δ_{norm} and $\mathsf{T}_{\mathsf{init}}$ we first select the less intensive function that is ad hoc secure with respect to δ_{norm}. Then we use the entire $\mathsf{T}_{\mathsf{init}}$ time to compute a chain from the last level, subject only to memory constraints, i.e., if memory exhaust before T_{init} we stop. For example, given σ_ℓ the length of the chain on the last layer we choose the number of levels ℓ. Then we set to 1 the length of each chain from level 1 to $\ell - 1$. In this way we minimize the memory and computational time for the upper layers. Since the number of upper layers is maximum, due to the reduced chain length, this also increases initialization overhead on the bus. $\mathsf{T}_{\mathsf{init}}$ should be only slightly overloaded since the initialization time is minimum when the number of levels is maximum. If memory is also exhausted by the first layer, we cut down as many elements as are needed to fit the upper layers.

The parameters of the scheme can be computed as follows. Having $\mathsf{T}_{\mathsf{run}} = \delta_{\mathsf{norm}} \cdot [2^{\ell^{\mathsf{GETA}}-1} \cdot (\sigma + 1) - 1]$ the number of levels and the disclosure delay are: $\ell^{\mathsf{GETA}} = \lceil \log_2 \mathsf{T}_{\mathsf{run}} + \delta_{\mathsf{norm}}/\delta_{\mathsf{norm}} \cdot (\sigma + 1) \rceil + 1$, $\delta_i^{\mathsf{GETA}} = \delta_{\mathsf{norm}} \cdot 2^{\ell^{\mathsf{GETA}}-i-1} \cdot (\sigma + 1)$.

D. Comparison and Limitations

We now give an explanatory example for the *ad hoc secure* schemes. Assume the output of MD5 can be truncated to 32 bits for ad hoc security with respect to a delay of 10^{-3} seconds, 48 bits for 1 second, 64 bits for 10^3 seconds and left unchanged to 128 bits for any delay greater than these. The computational timings are around 11×10^{-6} according to the experimental results from the next section and they serve here as a calibration example.

Let us fix for our example the disclosure delay to $\delta_{\mathsf{norm}} = 1\ \mathrm{ms}$ with a one year runtime of the protocol $\mathsf{T}_{\mathsf{run}} = 31 \times 10^6$ s and $\mathsf{T}_{\mathsf{init}} = 1$ s.

With the $GETA$ scheme, by using computational power as restriction we can compute up to 90.000 elements in $\mathsf{T}_{\mathsf{init}}$ which is far too much for the memory, so we limit σ_ℓ to 1000 which is reasonable to fit in memory and would ease the computation. This gives $\ell = 26$ with length 1000 on level 26 and 1 on the other levels. The disclosure delay on level 26 is $10^{-3}s$ so we can use the 32 bit version. On level 25 the disclosure delay is

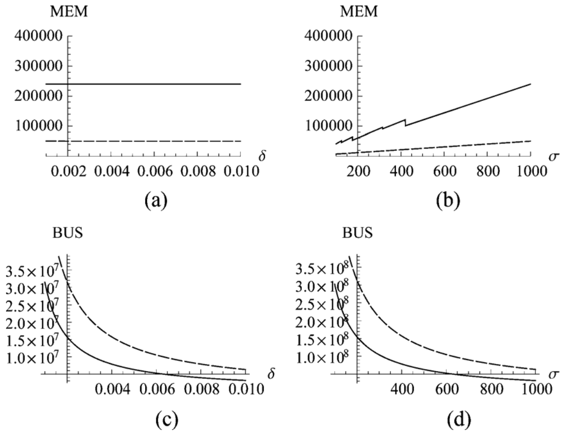

Fig. 4. Comparison between $BETA$ (continuous line) and $GETA$ (dotted line) schemes at MEM and BUS requirements for: fixed $\sigma = 1000$ and variable δ_{norm} in (i) and (iii), variable δ_{norm} and fixed $\sigma = 1000$ in (ii) and (iv).

around 2 s so we skip to the 48 bit version which can be used up to level 23 that has a delay of 8 s. Levels up to 16 can use the 64 bit version and the rest of the levels will use the 128 bit version. The overall memory load is $1000 \times 32 + 3 \times 48 + 7 \times 64 + 15 \times 128 = 34512$ bits. The commitments on the bus reach up to 11×10^9 bits in a year.

With the $BETA$ scheme, by fixing $\sigma = 1000$ we get $\ell = 4$ but this is due to the ceiling which increased the value from the actual 3.5. With chains of 1000 elements this will increase $\mathsf{T}_{\mathsf{run}}$ too much and will require too much memory as well. Note that the delay on level 4 is 10^{-3} and it increases on each level with a factor of 1000, which means that roughly level 1 uses the 128 bit version, level 2 the 64 bit version, level 3 the 48 bit version and level 4 the 32 bit version. For chains of length 1000 this gives a memory load of $1000\times32+1000\times48+1000\times64+1000\times128$, that is 272000 bits which is almost 8 times more than for the $GETA$ scheme but $\mathsf{T}_{\mathsf{run}}$ has also increased about 30 times. By empirical test we get that $\sigma = 428$ will not change $\mathsf{T}_{\mathsf{run}}$ and will require only 116416 which is just 3 times more than for $GETA$ scheme. For the commitments on the bus its the reverse as they reach up to 3.7×10^9 bits in a year which is 3 times less than for $GETA$. Having a lighter bus load should be preferable for practical applications, therefore $BETA$ seems to be better.

Fig. 4 shows the generic difference between the $BETA$ and $GETA$ schemes assuming fixed delay and variable length or the reverse. As already shown in the previous example, the relevant aspect is that $BETA$ gives a lighter bus due to fewer chain commitments. However, $GETA$ is much better in terms of memory requirements which is also a relevant constraint.

E. Security Considerations for the Ad Hoc Secure Scheme

It is not easy to define rigorous parameters for ad hoc security. Here we used 32 bits as the minimum recommended size for authentication tags against "real-time" attacks by the most recent ECRYPT guidelines on key sizes [30]. This security level is certainly enough against common equipments, such as standard CPU's, that can perform hashes in the order of 10^6 per second. Dedicated equipments, e.g., built on FPGAs, can reach billions of crypto-operations per second which is close to the 32 bit security bound. However, such equipments are not cheap, their cost is usually in the order of tens of thousands of dollars and thus they are not available to average adversaries. Even for the case of such adversaries, with dedicated hardware we believe that 48 bits should still be infeasible to invert in less than a second.

These considerations are helpful to assess the security of the protocol from a quantitative perspective. For validating the security of larger systems, where such protocols are part of, automatic verification with specialized tools is an alternative. Using such techniques has become more frequent in industrial systems, a case study on fieldbus is available in [12].

F. Synchronization Issues

Clock drifts between oscillators can lead to more frequent resynchronizations. Common recommendations for CAN bit timings are a minimum oscillator tolerance of 1.49% at 125 kbps and 0.49% at 1 Mbps. These tolerances are enough to eliminate frequent resynchronization for most disclosure delays. However, if we push the disclosure delays to their lower limit this can become a relevant issue.

We consider an example to clarify this. Around 300 μs is the minimum delay achieved in our application setting for a 1 Mbps CAN bus. A tolerance of 0.49% will result in a maximum of $2 \times 0.49\% \approx 1\%$ drift between the sender and the receiver (considering the worst scenario in which oscillators drift apart in opposite directions). This means that at each 300 μs the receiver clock drifts with 3 μs which means that after 100 packets the receiver will either drop all subsequent packets (if its clock is faster), or an adversary may be able to forge packets (if its clock is slower). However, such a clock drift results from using oscillator tolerances that are quite at the edge. Let us emphasize that in our practical setting the two Infineon TriCore controllers had a drift of around 2.73 seconds after 24 hours when running at maximum speed. This leads to a drift of 3.15 ns at each 100 μs and even after 1 second of broadcast the drift is around 3 μs which will keep the protocol secure if we set $\xi = 3\ \mu$s. Thus, the first and the most natural solution is to use better oscillators which are available on the market and already present in most of the devices. In case they are not available in a particular setting, higher authentication delays should be considered.

Nodes that have weak oscillator tolerances can send a resynchronization request by placing a nonce on the bus and the sender will answer with the usual $\mathrm{MAC}_{\mathsf{K}}(time, nonce)$ which will be authenticated with the forthcoming key. For security reasons, we enforce such a resynchronization to be performed each time it is expected that the clock of the sender and receiver drifted by ξ. In our practical scenario it leads to a resynchronization packet at around each minute (depending on the exact disclosure delay, safety margin and microcontroller speed). For efficiency, synchronization can be in this way processed for more than one receiver at once in the sense that different nonces from different receivers can be merged by the broadcast master in one response (care should be taken since this will increase the synchronization error while it is still mandatory for each receiver to send its own fresh nonce). The resynchronization procedure shouldn't raise scalability problems if it is not triggered too often.

TABLE II
PERFORMANCE OF S12X, XGATE AND TRICORE

Length (bytes)	S12X (ms)		XGATE (ms)		TriCore (ms)	
	MD5	SHA-256	MD5	SHA-256	MD5	SHA-256
0	0.732	5.51	0.313	3.155	0.010	0.042
26	0.739	5.49	0.318	3.145	0.011	0.042
62	1.414	10.86	0.605	6.24	0.018	0.081
80	1.374	10.80	0.592	6.22	0.019	0.081

G. Coexistence With Other Traffic

The influence of the broadcast traffic on already existent real-time CAN bus traffic and vice-versa is a relevant topic. Let us underline that from the authentication point of view, timing is critical only for keys and MAC codes (keys are disclosed at precise moments and MACs must be released no later than the corresponding key), while the authenticated messages can be released at any point. In the case when the broadcast authentication related traffic has higher priority than the real-time traffic existent on the bus, by construction of the equidistant timing protocol we assured that the keys are uniformly distributed over the life-time of the protocol. Thus, a scenario with periods of burst, with more keys than usual on the bus, will not occur. More, we enforce the use of the same priority for the MAC as for the message they authenticate and thus real-time constraints should not be violated. If the broadcast authentication protocol has lower priority than some of the existent traffic some tweaks in the protocol are possible to accommodate it with already existent higher priority traffic. For example MACs cannot be released later than the keys but their precise release time is not so critical, thus they can be released sooner. Keys however need to be released at the precise time they are scheduled or as soon as the bus is free afterwards (note that releasing the keys later doesn't cause any insecurity). Indeed, in the case of a heavily loaded bus, disturbances between different types of traffic are unavoidable. In such situations, physical separation between the two kinds of traffic may be the best alternative.

IV. EXPERIMENTAL RESULTS

A proof-of-concept implementation was done in order to determine the behaviour of the proposed solution in a real environment. We made tests on Freescale S12X (16-bit) and Infineon TriCore (32-bit), two microcontrollers frequently used in automotive and industrial applications. The test setup consists of a master node (which holds the key chains) and multiple slave nodes. We discuss next the experimental results.

A. Computational Performance

First, we evaluated the performance of the microcontrollers for computing cryptographic primitives. Table II holds the results of our measurements. The S12X results were obtained for a frequency of 40 MHz (when overclocking at 80 MHz the execution speed is doubled) while TriCore microcontrollers were running at 180 MHz. These measurements show that on average the primitives were performed approximately 2.12 times faster on XGATE than on S12X while the TriCore implementations are, as expected, much faster (1 to almost 2 orders of magnitude).

TABLE III
SOME PARAMETERS CHOICES FOR THE INFINEON TRICORE PLATFORM

Key size	3 levels			3 levels		
(bytes)	δ(*ms*)	M (bytes)	σ	δ(*ms*)	M (bytes)	σ
4	10	37704	3142	1	81216	6768
mixed	10	75408	3142	1	162432	6768
16	10	150816	3142	1	324864	6768

Key size	3 levels			4 levels		
(bytes)	δ(*μs*)	M (bytes)	σ	δ(*μs*)	M (bytes)	σ
4	315.24	120024	10002	329.44	15840	990
mixed	318.08	239304	9971	332.28	39520	988
16	471.44	419808	8746	479.96	57664	901

The computation speed, memory and the low speed CAN transceiver offer a considerable bound to the communication speed achievable on the S12X implementation. The overall computation time can be decreased by computing all cryptographic primitives on the XGATE co-processor. While the cryptographic functions are being computed on XGATE, the main CPU is free to execute other tasks, such as receiving messages or sending messages that have been already built. After the program implementation, the total RAM memory left for storing key chains can hold 1216 elements (16 bytes each). Having this upper limit for MEM, ℓ and σ have to be determined for best performances based on the bus speed and the wanted disclosure delay. If we consider packets of 16 bytes in size the measured bus speed for sending authenticated packets is 578 packets/second (one packet each 1.73 ms) which could be considered too slow in a time critical system.

B. Adjusting Parameters

Choosing the best combination of parameters highly depends on the application and on the devices used for implementation. The parameters were chosen so that they fit for authenticated communication over a time span of 10 years without the need of reinitialization. Table III contains some of the parameter combinations we tried for our TriCore implementation. The aim was to find the best parameter combinations to obtain small authentication delays (equidistant timing was used in all variants). The key chains were computed using MD5 and for the MACs we used HMAC-MD5. We underline that although MD5 does not offer collision resistance anymore it is still secure enough for our application that requires only secondary pre-image resistance. Using stronger hash functions from the SHA-2 family or from the SHA-3 candidates will impair performances without much practical justification. As computational results for these functions are available in Table II, the protocol performance can be easily deduced for these cases as well. For the case of the S12X microcontrollers, due to the reduced computational power and mostly due to the bus speed reduced to 125 kbps, delays in the order of 10 ms were the best we could achieve.

In the case of the Infineon TriCore, as the greatest communication overhead was caused by the CAN frame format and maximum transfer speed, we tried different key sizes. When using the whole 16 bytes of the MD5 generated key, the smallest authentication delay we could obtain was 471.44 μs for 3 levels. By using smaller keys (which we obtained by truncating the 16 bytes output of MD5), the communication overhead is reduced at the cost of a lower security level. A 4 byte key used on all levels would enable an authentication delay of 315.24 μs for a 3 level setup and 340.8 μs for 5 levels. A better trade-off can be made by assigning different length keys to each level in order to provide enough security for the key lifetime. As an example, we assigned a 4 byte key to the first level, an 8 byte key for the second level, 12 bytes for the third level and 16 bytes for all the other levels. For a 5 level setup, each 4 byte key will have to last for 343.64 μs while the 8 byte keys on the second level will have a life time of 84.88 ms. We underline that these delays show the minimum achievable with our implementation and, since they are on the extreme side, reaching them is quite consuming for the computational resources of the devices as well as for the bus load. For practical settings, delays from 1 to 10 ms should be easily handled by the TriCore controllers and will result in a clean deployment without consuming much of the controller's resources or communication bandwidth.

V. CONCLUSIONS

We studied different trade-offs for the schemes in order to determine an optimal choice of parameters. As expected, the TriCore microcontroller clearly outperformed the S12X and the authentication delays dropped by almost two orders of magnitude. To improve even more, as the main limitation was the speed of the bus, the ad hoc scheme was able to get the delays even lower. In the best case these delays were in the order of several hundreds micro-seconds. By this research we hope that we give a first analysis on the feasibility of using cryptographic authentication in CAN networks. Since by the nature of this family of authentication protocols small delays cannot be avoided, some applications may be too restrictive for this approach. But we do expect that the delays achieved here are suitable in many practical real-time scenarios. Integrating the broadcast authentication protocol in such a real world scenario is of great interest as future work for us.

ACKNOWLEDGMENT

The authors thank the reviewers for comments and suggestions that helped them to improve their work.

REFERENCES

[1] L. Almeida, P. Pedreiras, and J. A. G. Fonseca, "The FTT-CAN protocol: Why and how," *IEEE Trans. Ind. Electron.*, vol. 49, no. 6, pp. 1189–1201, 2002.
[2] H. K. Aslan, "A hybrid scheme for multicast authentication over lossy networks," *Comput. Security*, vol. 23, no. 8, pp. 705–713, 2004.
[3] H. Bar-El, "Intra-vehicle information security framework," in *Proc. 9th Embedded Security in Cars Conf., ESCAR*, Sep. 2009.
[4] M. Barranco, J. Proenza, and L. Almeida, "Quantitative comparison of the error-containment capabilities of a bus and a star topology in CAN networks," *IEEE Trans. Ind. Electron.*, vol. 58, no. 3, pp. 802–813, 2011.
[5] M. Barranco, J. Proenza, G. Rodriguez-Navas, and L. Almeida, "An active star topology for improving fault confinement in CAN networks," *IEEE Trans. Ind. Electron.*, vol. 2, no. 2, pp. 78–85, May 2006.
[6] D. Berbecaru, L. Albertalli, and A. Lioy, "The ForwardDiffSig scheme for multicast authentication," *IEEE/ACM Trans. Netw.*, vol. 18, pp. 1855–1868, Dec. 2010.
[7] F. Bergadano, D. Cavagnino, and B. Crispo, "Individual authentication in multiparty communications," *Comput. Security*, vol. 21, no. 8, pp. 719–735, 2002.

[8] G. Cena and A. Valenzano, "An improved CAN fieldbus for industrial applications," *IEEE Trans. Ind. Electron.*, vol. 44, no. 4, pp. 553–564, 1997.

[9] G. Cena and A. Valenzano, "FastCAN: A high-performance enhanced CAN-like network," *IEEE Trans. Ind. Electron.*, vol. 47, no. 4, pp. 951–963, 2000.

[10] Y. Challal, A. Bouabdallah, and H. Bettahar, "H2A: Hybrid hash-chaining scheme for adaptive multicast source authentication of media-streaming," *Comput. Security*, vol. 24, no. 1, pp. 57–68, 2005.

[11] S. Checkoway, D. McCoy, B. Kantor, D. Anderson, H. Shacham, S. Savage, K. Koscher, A. Czeskis, F. Roesner, and T. Kohno, "Comprehensive experimental analyses of automotive attack surfaces," *Proc. 20th USENIX Conf. on Security*, p. 6, 2011.

[12] M. Cheminod, A. Pironti, and R. Sisto, "Formal vulnerability analysis of a security system for remote fieldbus access," *IEEE Trans. Ind. Informat.*, vol. 7, no. 1, pp. 30–40, Feb. 2011.

[13] D. Dzung, M. Naedele, T. P. V. Hoff, and M. Crevatin, "Security for industrial communication systems," *Proc. IEEE*, vol. 93, no. 6, pp. 1152–1177, Feb. 2005.

[14] N. Falliere, L. O. Murchu, and E. Chien, W32.Stuxnet Dossier Symantec, 2011.

[15] W. Granzer, F. Praus, and W. Kastner, "Security in building automation systems," *IEEE Trans. Ind. Electron.*, vol. 57, no. 11, pp. 3622–3630, Nov. 2010.

[16] B. Groza and P.-S. Murvay, "Higher layer authentication for broadcast in controller area networks," in *Proc. Int. Conf. Security Cryptography (SECRYPT)*, 2011, p. 72.

[17] H. Hartenstein and K. Laberteaux, *VANET Vehicular Applications and Inter-Networking Technologies*. New York, NY, USA: Wiley, 2009.

[18] K. Koscher, A. Czeskis, F. Roesner, S. Patel, T. Kohno, S. Checkoway, D. McCoy, B. Kantor, D. Anderson, H. Shacham, and S. Savage, "Experimental security analysis of a modern automobile," in *Proc. Security Privacy (SP), 2010 IEEE Symp.*, May 2010, pp. 447–462.

[19] L. Lamport, "Password authentication with insecure communication," *Commun. ACM*, vol. 24, pp. 770–772, Nov. 1981.

[20] K. Lemke, C. Paar, and M. Wolf, *Embedded Security in Cars Securing Current and Future Automotive IT Applications*. New York, NY, USA: Springer Verlag, 2006.

[21] D. Liu and P. Ning, "Efficient distribution of key chain commitments for broadcast authentication in distributed sensor networks," in *Proc. 10th Ann. Netw. Distributed System Security Symp.*, 2003, pp. 263–276.

[22] D. Liu and P. Ning, "Multilevel μtesla: Broadcast authentication for distributed sensor networks," *ACM Trans. Embed. Comput. Syst.*, vol. 3, pp. 800–836, Nov. 2004.

[23] P. Martí, A. Camacho, M. Velasco, and M. El Mongi Ben Gaid, "Runtime allocation of optional control jobs to a set of CAN-based networked control systems," *IEEE Trans. Ind. Informat.*, vol. 6, no. 4, pp. 503–520, Nov. 2010.

[24] R. C. Merkle, "A digital signature based on a conventional encryption function," in *Proc. Conf. Theory Appl. Cryptographic Techniques on Advances in Cryptology, CRYPTO '87 Springer-Verlag*, London, U.K., 1988, pp. 369–378.

[25] R. Mitchell, "Tutorial: Introducing the XGATE Module to Consumer and Industrial Application Developers," Freescale, 2004.

[26] J. Munoz-Castaner, R. Asorey-Cacheda, F. J. Gil-Castineira, F. J. Gonzalez-Castano, and P. S. Rodriguez-Hernandez, "A review of aeronautical electronics and its parallelism with automotive electronics," *IEEE Trans. Ind. Electron.*, vol. 58, no. 7, pp. 3090–3100, Jul. 2011.

[27] A. Perrig, R. Canetti, D. Song, and J. D. Tygar, "Efficient and secure source authentication for multicast," in *Proc. Network Distributed System Security Symp., NDSS '01*, 2001, pp. 35–46.

[28] A. Perrig, R. Canetti, D. Song, and J. D. Tygar, "Spins: Security protocols for sensor networks," in *Proc. 7th Ann. ACM Int. Conf. Mobile Computing Networks (MobiCom 2001)*, 2001, pp. 189–199.

[29] A. Perrig, R. Canetti, J. Tygar, and D. X. Song, "Efficient authentication and signing of multicast streams over lossy channels," in *Proc. IEEE Symp. Security Privacy*, 2000, pp. 56–73.

[30] ECRYPT II, Yearly Rep. on Algorithms and Keysizes (2009–2010), Revision 1.0 2011.

Bogdan Groza (M'13) received the Dipl.Ing. and Ph.D. degrees from Politehnica University of Timisoara (UPT), Romania, in 2004 and 2008, respectively.

He is currently a lecturer at the Faculty of Automatics and Computers (UPT), Politehnica University of Timisoara, Romania. His research interests are in the field of information security and cryptography with focus on provable security, authentication protocols and applied cryptography in embedded and industrial systems. Since 2004, he has directed and participated in several national and international research projects in this field.

Stefan Murvay (S'12) received the Dipl.Ing. and M.Eng. degrees in 2008 and 2010, respectively, both from Politehnica University of Timisoara, Romania. He is currently enrolled as Ph.D. student at the Faculty of Automatics and Computers in the same institution.

His research interest is in information security and embedded systems.