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. As bandwidth proved to be the main limitation, to provide clear bounds on the effectiveness of the solution we used low speed, fault tolerant and high speed communication as well.**

Index Terms— **Authentication, Broadcast, Controller Area Network, 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 [32], the current specifications for CAN are found in the newer standard ISO-11898 [34]. Numerous further improvements appeared in the literature: dual channel architectures [10], flexible time-triggered communication [1], star topologies [6], [5], dynamic identifiers to improve timing requirements [9], etc. It's success in vehicle applications led to the introduction of CAN in the aeronautical sector with several domain specific adaptations [28]. As data throughput has increased inside cars, high performance buses were developed in the last decade, e.g., FlexRay, but because of its efficiency and reduced cost CAN is still present in most automotives produced today. Due to its reliability and excellent price-performance ratio, CAN bus is likely to remain wide-spread for a long time, especially in applications that do not require high bandwidth.

Manuscript received September 20, 2011. Accepted for publication November 29, 2012. Copyright © 2012 IEEE. Personal use of this material is permitted. However, permission to use this material for any other purposes must be obtained from the IEEE by sending a request to pubs-permissions@ieee.org.
B. Groza and S. Murvay are with the Faculty of Automatics and Computers, Politehnica University of Timisoara, Romania. Phone: +40-256-403242, e-mail: bogdan.groza@aut.upt.ro, stefan.murvay@gmail.com

Reliability was always a main concern in control systems and in automotives in particular, but only with respect to natural phenomenons (electromagnetic disturbances, thermal noise, etc.) or accidents of various causes and not in front of active Dolev-Yao adversaries. For this purpose, CAN has been designed to deal with errors and to recover afterwards. The probability of an undetected error on CAN is extremely low, informally one undetected error occurs at about one thousand years for each vehicle that operates eight hours a day with an error each 0.7s. The first in-depth study on the performance of CAN error detection mechanisms was done by Charzinski in [12].

In the last decade, industrial control and automation systems become opened to malicious adversaries and a significant part of the security community focused on alleviating potential threats in such environments [15], [17]. 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 [16]. As for in-vehicle security, recent research showed current automobiles to be unexpectedly vulnerable to external adversaries [20], [13] 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 [22] and [19], both contain relevant chapters on intra-vehicle security. Also several research papers published in the last years address these issues [4]. Still, to best of our knowledge there is no implementation available to assure authenticity in CAN networks.

In this context, of malicious adversaries that can manipulate messages over the network, CAN does not have intrinsic support for any kind of security. Indeed, such kind of security is not needed if one sees CAN as operating in a secure perimeter. But, it is very likely that soon CAN-like networks will operate in environments that are open for intruders. To withstand such threats, security can be implemented at the application level. In a different context, such improvements were used in the past when deterministic delays were needed on CAN bus with the development of Time Triggered CAN [35]. Thus, the main intention of this paper is to develop a higher layer implementation of a broadcast authentication protocol and to study several trade-offs to increase its efficiency.

A. Related work on secure broadcast protocols

Although digital signatures provide an elegant method for signing broadcast data, they are not the solution in our context because of both the computational and communication overhead. As messages are short in CAN networks, usually fitting in the 64 bits of data carried by one CAN frame, using a public-key primitive such as the RSA requires thousands of bits and causes a significant 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 in [26]. Although they are frequently mentioned in the literature as a cheaper alternative to conventional signatures, they are quite unused in practice, mostly because of their one-time nature. Using Merkle trees makes them viable for multiple uses, but it requires sending an entire path of a tree, and generating, potentially storing all this tree on the signer side, which requires even more resources. There is good literature available on the subject but this line of research appears to have a reduced practical impact as one-time signatures are not so common in practice.

In contrast, symmetric primitives were efficiently employed for authentication in constrained environments such as sensor networks [30], [23], [24]. Due to the broadcast nature of CAN, protocols similar to the well known TESLA protocol [31], [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 [25].

There is an extensive bibliography related to the TESLA protocol. Its history can be traced back to Lamport's scheme which uses one-way chains to authenticate users over an insecure network [21]. The work of Bergadano et al. [8] proposes several variants of one-way chain based protocols, with or without time synchronization. Previous work which inspired this family of protocols is the Guy Fawkes protocol from [2]. The context which is more related to our setting here is that of the application of such protocols in sensor networks. In particular, several trade-offs for sensor networks were studied by Liu and Ning in [23], [24] and several variants of the protocols are presented by Perrig et al. as well in [31], [29].

Also several papers addressed hybrid versions in which asymmetric primitives are mixed with key-chains in order to obtain trade-offs [7], [11], [3]. However, these variants have a bigger communication or computational overhead and do not appear to be appropriate for our application setting.

B. 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, there is no result so far, to best of our knowledge, 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 [18] 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. More concrete, while our previous result allowed an authentication delay of around 10*ms*, we make the authentication delay here to drop in the order of 300μs.

The improvements provided here are relevant as the authentication delay is critical for control scenarios. This is different to the usual sensor-network scenario where TESLA like protocols are frequently used because in sensor networks other constraints are more prevalent. For example, energy consumption is a critical issue in sensor networks, but usually for control units inside a car this is not a main concern since controllers do not strongly rely on small batteries. The most critical part, in control systems where this protocol is mostly used, is the authentication delay, i.e., how fast a packet can be deemed as authentic. In particular we must assure that a node, if the bus is not taken by a higher priority message, is able to transmit the message and the message can be checked for authenticity as soon as possible. This condition is initially limited by the computational power, but as checking for authenticity can happen only as soon as the disclosure delay expires and the next element of the chain is committed, this also depends on the structure of the chain which is determined by the amount of memory, and also by the bandwidth. 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.

The paper is organized as follows. Section 2 gives an overview of the protocol, starting from several details of the CAN protocol to the examination of the influence of chain lengths, structure and timings on performance. In section 3 we present some practical variants of the scheme and we discuss their theoretical efficiency as well as issues regarding synchronization and coexistence with other traffic on the bus. Experimental results on a state-of-the-art TriCore microcontroller from Infineon are provided in section 4. Section 5 holds the conclusions of our paper.

II. Environment and Protocol Description

CAN is a two wire broadcast bus as shown in Figure 1. There is also active interest toward using star topologies in CAN networks [6], [5] and this would have certain advantages from a security perspective. 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. Additionally 1 bit marks the start of frame and 7 bits mark its end. This structure is suggested in Figure 2. Arbitration is based on the identifier ID which has 29 bits in extended frames and 11 bits in standard frames. The winner is determined based on the state of a particular bit, namely recessive bits (value 1) are overwritten by dominant bits (value 0). So, if the case, all nodes can start to write a message at the same moment on the bus, but, whenever a node writes a recessive bit and reads a dominant one it means that it lost the arbitration and will stop, otherwise it can continue. 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. The identifiers are generally fixed and are not changed during runtime, however assigning dynamic IDs was suggested in the past [9] in order to improve on timing requirements and can be considered in further optimizations for particular scenarios. After each 6 consecutive bits of identical values a stuffing bit of different value is added. The body of a message can have at most 8 bytes and is followed by a 15 bit CRC. In the worst case a frame can have 154 bits out of which only 64 bits are of actual useful information. Thus, the overhead is high from the basic design of the protocol, in the worst case exceeding 50%. But this is needed to achieve reliability as mentioned before. Two kinds of CAN nodes are commonly available on the market: fault tolerant low-speed nodes which operate at 125kbps and high-speed nodes that work up to 1Mbps.

Indeed, as shown by this brief description, 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 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, the design paradigm is the following. Memory, computational speed, bandwidth, initialization time

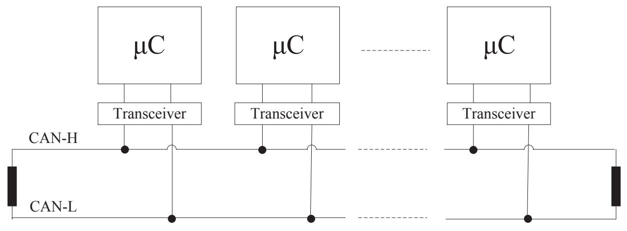

Fig. 1. Generic CAN topology.

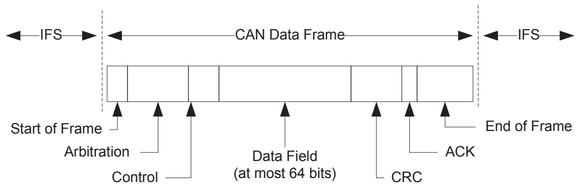

Fig. 2. Structure of a CAN data frame.

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. To improve on this delay, we design several variants of the protocol that are presented in section 3.

All protocol variants use multiple levels of one-way key chains with the structure suggested in Figure 3. The relevant notations with respect to the chain structure are: ℓ the number of chain levels, $\sigma_i, i = 1..\ell$ the length of the chain on level i, $\delta_i, i = 1..\ell$ the disclosure delay on level i, ξ the safety margin for releasing authentication packets and δ_{norm} the *normalized disclosure delay* which will be clarified in the next section along with other details. Informally, bullets depict keys from the key chains and the horizontal black arrows denote that they are derived from a previous key. As usual in such protocols, 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 communication bus are depicted as well, the dotted lines from an element of the chain to the packet denotes that the element was used as a key, and for the re-initialization packets in particular 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. For example, each principal can authenticate itself by using a public key certificate that is signed by a trusted authority. Initial authentication based on public-key infrastructure can be preferable to assure composability. Thus we require that each node must store the public key of a trusted authority. 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. This is usually achieved in two protocol steps as follows: $A \rightarrow B : N_A; B \rightarrow A : Sig_B(t_B, N_A)$. Here N_A denotes a nonce generated by principal A and t_B denotes the current time at principal B when sending its response. Afterwards, the round trip time ε_{AB} becomes the synchronization error. If the nonce was sent by A at time t_0 and now A's clock points to t_1 then A knows that the time on B side is in the interval $[t_B + t_1 - t_0,\ t_B + t_1 - t_0 + \varepsilon_{AB}]$. However, in our scenario a digital signature costs tens, or hundreds of milliseconds, which will result in an even larger disclosure delay. Because of this, instead of a digital signature we will use a message authentication code which is several orders of magnitude faster. In particular, in our experiments, the round-trip-time was less in the order of several hundreds micro-seconds as shown in the section dedicated to experimental results.

A. Sender's Perspective

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., $\mathbf{T}_\ell = \{(\sigma_1, \delta_1), (\sigma_2, \delta_2), ..., (\sigma_\ell, \delta_\ell)\}$.

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., $\mathbf{F}_\ell = \{\mathsf{F}_1, ..., \mathsf{F}_\ell\}$.

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 τ_{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 Figure 3.

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 t_{start} the disclosure time of K_τ is: $\mathcal{DT}(\mathsf{K}_\tau, \mathbf{T}_\ell) = t_{start} + \sum_{i=1..\ell}(\delta_i \cdot \tau_i)$.

B. Receiver's Perspective

We consider the case of a receiver $\mathcal{R}$ and sender $\mathcal{S}$ with synchronization error $\varepsilon_{\mathcal{S},\mathcal{R}}$. 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 $\varepsilon_{\mathcal{S},\mathcal{R}}$ 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: $\mathcal{ET}_{\min}(t_\mathcal{R}) = t_\mathcal{R} - t_{sync} + t_\mathcal{S}$, $\mathcal{ET}_{\max}(t_\mathcal{R}) = t_\mathcal{R} - t_{sync} + t_\mathcal{S} + \varepsilon_{\mathcal{S},\mathcal{R}}$.

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.

C. Generic Description of the Protocol

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 $\varepsilon_{\mathcal{S},\mathcal{R}}$, 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

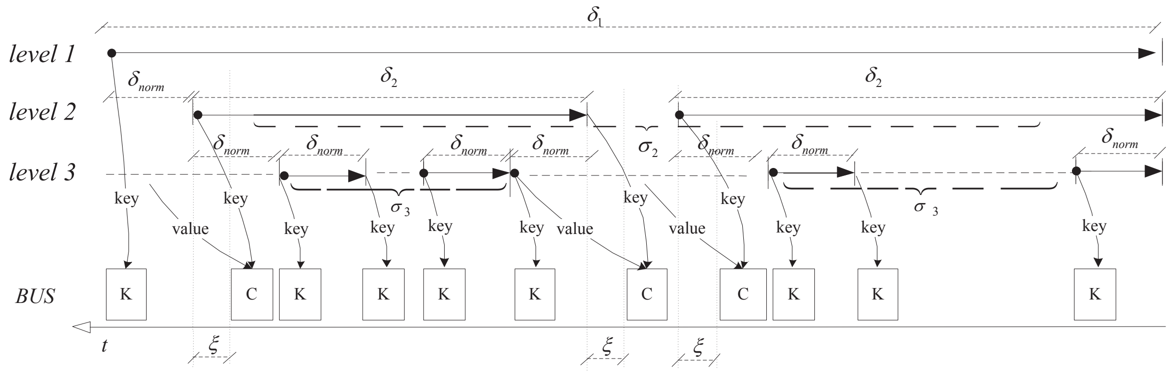

Fig. 3. Broadcast sequence with normalized time δ_{norm}.

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 [8] and [31]. 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. A more formal proof sketch can be done by using random oracles. It is commonly acknowledged that although random oracles do not give an absolute proof they can be used at least as a sanity check to prove the security of protocols. If we assume that oracle O^{F} that computes function F can be replaced by a random oracle O^R, which outputs k bits, the proof is straight forward. Assume that the adversary has witness polynomially many queries $p(k)$ to oracle O^R. Suppose at some point the adversary is forced to produce $\mathrm{MAC}_{\mathcal{K}}(M_{\mathrm{Adv}})$ for some message of its choice. The adversary knows just $O^R(\mathcal{K})$ which is the output of the random oracle and $\mathcal{K}$ is unpredictable subject to the fact that it may have been already asked by the adversary to O^R. This means he can guess it and produce a valid MAC only with probability $1/(2^k - p(k))$ - which is negligible.

D. 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 $\mathsf{T}_{\mathsf{init}}$ 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 $\mathsf{CPU}_{\mathsf{init}}$ refers to the amount of work during initialization and $\mathsf{CPU}_{\mathsf{run}}$ during runtime. By $\mathsf{MEM}^{\mathsf{total}}$, $\mathsf{CPU}^{\mathsf{total}}$ and $\mathsf{BUS}^{\mathsf{total}}$ 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 $\parallel \mathbb{K}_{\mathbf{T},\mathbf{F}}[i] \parallel$ denote the total number of keys on level i disclosed during protocol lifetime and $\langle\!\langle \mathbb{K}_{\mathbf{T},\mathbf{F}}[i] \rangle\!\rangle$ the number of key chains from level i.

Obviously we have $\parallel \mathbb{K}_{\mathbf{T},\mathbf{F}}[i] \parallel = \sigma_i \cdot \langle\!\langle \mathbb{K}_{\mathbf{T},\mathbf{F}}[i] \rangle\!\rangle$ 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.

Definition 9. Let $\{(c_0,m_0),(c_1,m_1),...,(c_\ell,m_\ell)\}$ 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 caused by key-chains:

$$\mathsf{MEM}^{\mathsf{key}}_{\mathsf{init}} = \mathsf{MEM}^{\mathsf{key}}_{\mathsf{run}} = \sum_{i=1,\ell} \sigma_i \cdot m_i \tag{1}$$

$$\mathsf{CPU}^{\mathsf{key}}_{\mathsf{init}} = \sum_{i=1,\ell} \sigma_i \cdot c_i \quad \mathsf{CPU}^{\mathsf{key}}_{\mathsf{run}} = \sum_{i=2,\ell} c_i \cdot (\parallel \mathbb{K}_{\mathbf{T},\mathbf{F}}[i] \parallel - \sigma_i) \tag{2}$$

$$\mathsf{CPU}^{\mathsf{com}}_{\mathsf{init}} = \sum_{i=1,\ell} c_i \quad \mathsf{CPU}^{\mathsf{com}}_{\mathsf{run}} = \sum_{i=2,\ell} c_i \cdot (\langle\!\langle \mathbb{K}_{\mathbf{T},\mathbf{F}}[i] \rangle\!\rangle - 1) \tag{3}$$

$$\mathsf{BUS}^{\mathsf{key}}_{\mathsf{run}} = \sum_{i=1,\ell} m_i \cdot \parallel \mathbb{K}_{\mathbf{T},\mathbf{F}}[i] \parallel \tag{4}$$

$$\mathsf{BUS}^{\mathsf{com}}_{\mathsf{init}} = \sum_{i=1,\ell} m_i \quad \mathsf{BUS}^{\mathsf{com}}_{\mathsf{run}} = \sum_{i=2,\ell} m_i \cdot (\langle\!\langle \mathbb{K}_{\mathbf{T},\mathbf{F}}[i] \rangle\!\rangle - 1) \tag{5}$$

Equation 1 gives the memory requirements which is the sum of the lengths of the chains. 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. Equation 2 gives computational time required for keys at runtime and initialization. In the initialization one chain is computed on each level. At runtime, there are $\langle\!\langle \mathbb{K}_{\mathbf{T},\mathbf{F}}[i] \rangle\!\rangle$ key-chains on each level, and each of them has to be computed except the first one which was computed during initialization which gives $\mathsf{CPU}^{\mathsf{key}}_{\mathsf{run}} = \sum_{i=2,\ell} c_i \cdot \sigma_i \cdot (\langle\!\langle \mathbb{K}_{\mathbf{T},\mathbf{F}}[i] \rangle\!\rangle - 1)$. Replacing $\sigma_i \cdot \langle\!\langle \mathbb{K}_{\mathbf{T},\mathbf{F}}[i] \rangle\!\rangle$ with $\parallel \mathbb{K}_{\mathbf{T},\mathbf{F}}[i] \parallel$ we get the claimed number of keys computed at runtime. In Equation 3 commitments are measured: one commitment on each level during initialization, and later for each chain on each level (except for the first one which was committed during initialization) one commitment is needed. Bus requirements for keys during runtime is given in Equation 4. All keys from all levels are sent on the bus, while there are no keys (just commitments) sent during initialization. Commitments are given in Equation 5. 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.

Definition 10. Given $\mathsf{RES} \in \{\mathsf{MEM}, \mathsf{CPU}, \mathsf{BUS}\}$, $\mathsf{state} \in \{\mathsf{run}, \mathsf{init}\}$ we define the protocol overheads as: $\mathsf{RES}^{\mathsf{load}}_{\mathsf{state}} = (\mathsf{RES}^{\mathsf{key}}_{\mathsf{state}} + \mathsf{RES}^{\mathsf{com}}_{\mathsf{state}})/\mathsf{RES}^{\mathsf{total}}_{\mathsf{state}}$.

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 $\mathsf{T}_{\mathsf{run}}$ 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 (Delayed) 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 on the last level and then compute the disclosure delays on the upper levels. These disclosure timings are suggested in Figure 3.

Definition 11. For the ETA-CAN we define the disclosure delays as: $\delta_\ell = \delta_{\mathsf{norm}} = \mathsf{T}_{\mathsf{run}} / \sum_{i=1,\ell} \parallel \mathbb{K}_{\mathbf{T},\mathbf{F}}[i] \parallel = \mathsf{T}_{\mathsf{run}} / \prod_{i=1,\ell} (\sigma_i + 1) - 1$, $\delta_i = \delta_{\mathsf{norm}} \cdot \prod_{j=i+1,\ell} (\sigma_j + 1), 1 \le i < \ell$.

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.

C. Balanced Equidistant Timing delayed Authenticated CAN (BETA-CAN)

Based on the previous remark on efficiency we explore the variant with chains of equal sizes on all levels. To clarify previous notations Figure 4 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: $\ell^{\mathsf{BETA}} = \left\lceil \log_{\sigma+1} \left(\frac{\mathsf{T}_{\mathsf{run}}}{\delta_{\mathsf{norm}}} + 1 \right) \right\rceil$. The disclosure delay of the last level is

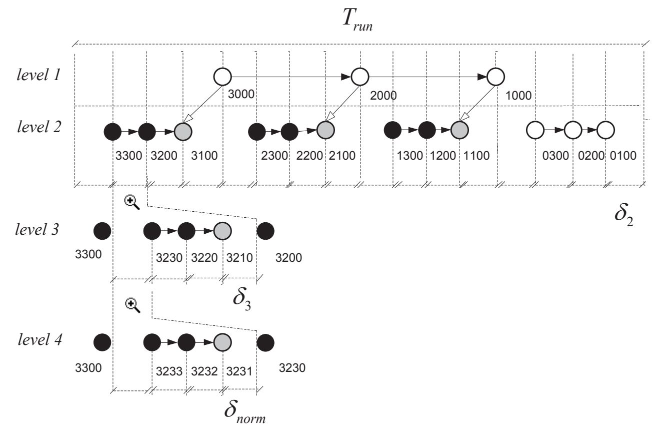

Fig. 4. Chain structure with *BETA* with $\ell = 4, \sigma_{\mathsf{norm}} = 3, \delta_{\mathsf{norm}} = \mathsf{T}_{\mathsf{run}}/255$.

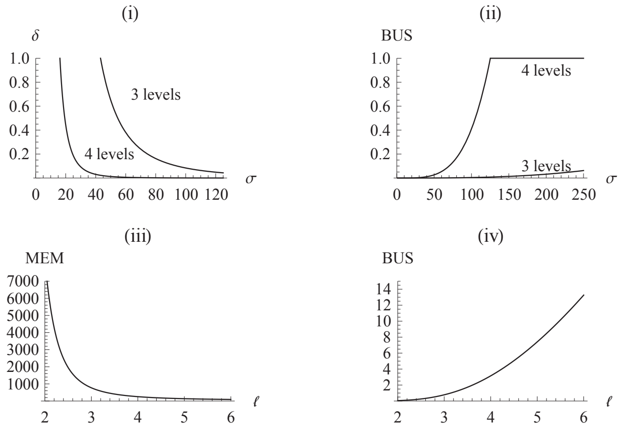

Fig. 5. Various overheads and delay variation with length or levels: (i) disclosure delay , (ii) overhead caused by keys and commitments on the bus, (iii) keys stored in memory and (iv) commitments on the bus (per second).

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.

Figure 5 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.

D. *Ad hoc secure Balanced levels ETA-CAN (Ad-BETA-CAN)*

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. More concrete, 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, 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.

E. *Ad hoc secure Greedy last level ETA-CAN (Ad-GETA-CAN)*

This 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

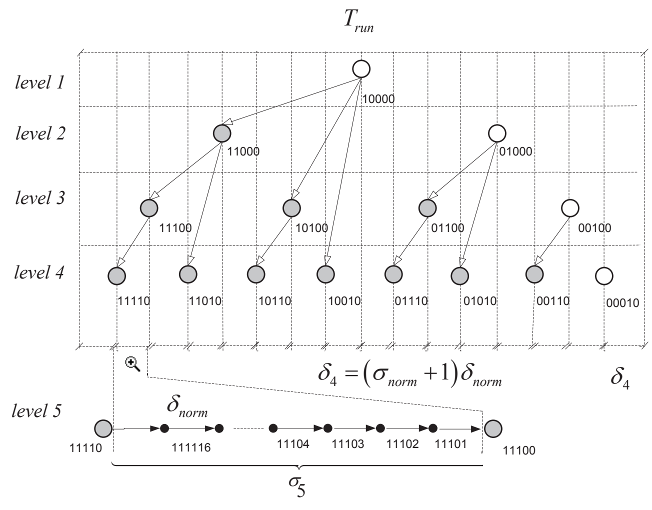

Fig. 6. Chain structure with *GETA* with $\ell = 5, \sigma_{\mathsf{norm}} = 16, \delta_{\mathsf{norm}} = \mathsf{T}_{\mathsf{run}}/255$.

TABLE I
SOME OVERHEADS AT INITIALIZATION AND RUN-TIME

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

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.

Figure 6 depicts the structure of the chains in this case while an explanatory example follows in the next subsection. 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: $\ell^{\mathsf{GETA}} = \left\lceil \log_2 \frac{\mathsf{T}_{\mathsf{run}} + \delta_{\mathsf{norm}}}{\delta_{\mathsf{norm}} \cdot (\sigma+1)} \right\rceil + 1$, $\delta_i^{\mathsf{GETA}} = \delta_{\mathsf{norm}} \cdot 2^{\ell^{\mathsf{GETA}}-i-1} \cdot (\sigma + 1)$.

Note that the same relations hold for the GETA scheme with or without ad hoc security. For simplicity, in the discussion that follows we will omit the prefix *ad* whenever it is clear from the context whether we refer to the standard scheme or to the ad hoc secure scheme.

F. 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. Note that computational speed is the same for all truncated versions and the gain is only in the bus load and memory requirements.

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

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 around $2s$ so we skip to the 48 bit version which can be used up to level 23 that has a delay of $8s$. 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 $1000 \times 32 + 1000 \times 48 + 1000 \times 64 + 1000 \times 128$, 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.

Figure 7 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, also a relevant constraint of our environment.

G. 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

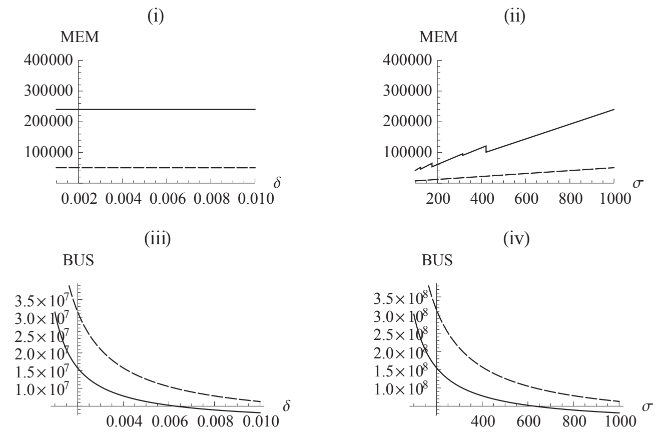

Fig. 7. Comparison between *BETA* (continuous line) and *GETA* (dotted line) schemes at MEM and BUS requirements for: fixed $\sigma = 1000$ and variable δ_{norm} in (i) and (iii), variable δ_{norm} and fixed $\sigma = 1000$ in (ii) and (iv).

for authentication tags against "real-time" attacks by the most recent ECRYPT recommendations on key sizes [33]. 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, built on FPGAs and more recently on GPU accelerators, can reach billions, i.e., 10^9, 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 and high financial resources, 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. Further, for validating the security of large scale systems, where such protocols can be 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 [14].

H. Synchronization issues

The synchronization error achieved by the loose time synchronization mechanism is small enough to accommodate the disclosure delays of the protocol. Even for the smallest disclosure delay in our practical implementation this error is still less than half of it. Indeed, this can be further improved by using more specialized protocols. A good start point could be the Precision Time Protocol (PTP) but this protocol is not cryptographically secure so further developments would be necessary.

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 $300\mu s$ is the minimum delay achieved in our application setting for a 1Mbps 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 $300\mu s$ the receiver clock drifts with $3\mu s$ 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 $3.15ns$ at each $100\mu s$ and even after 1 second of broadcast the drift is around $3\mu s$ 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.

Nevertheless, resynchronization can and should be performed in a more efficient manner by using the broadcast protocol as long as the node is not yet completely desynchronized. Thus, nodes that have weak oscillator tolerances must send a resynchronization request by placing a nonce on the bus and the sender will answer with the usual $\text{MAC}_{\mathsf{K}}(time, nonce)$ which will be authenticated with the forthcoming key K released in the broadcast protocol. More concrete, for security reasons, we enforce such a resynchronization to be performed when it is expected that the clock of the sender and receiver drifted by ξ. This leads in our practical scenario to send a resynchronization packet in the worst case at each minute (depending on the exact disclosure delay, safety margin and microcontroller speed). Also note that 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.

I. 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. First, 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. There are two cases that need to be considered.

In the first case, the broadcast authentication traffic has higher priority than the real time real-time traffic existent on the bus (which can be disturbed in this way). By construction of the equidistant timing protocol, we assure 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. The only burst may be due to MACs that also need to be sent before the keys are disclosed. To avoid disturbance of the real time traffic on CAN, we enforce the use of the same priority for the MAC as for the message they authenticate, and not a higher one. Thus, real-time constraints will not be violated - each MAC has the same priority as the message it authenticates.

In the second case, the broadcast authentication protocol has lower priority than some of the existent traffic. Indeed, it is not our main intention to use the protocol in such scenarios, but tweaks in the protocol are possible to accommodate it with already existent higher priority traffic. For this case we propose the following strategy concerning the keys and the MACs. As MACs cannot be released later than the keys but their precise release time is not critical, they will be released as quick as possible (roughly short after the release of the previous key) which gives higher chances for a MAC to be released in time. For the keys however, as premature release is not feasible since this will change the timings of the scheme, they will 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. An exception is the ad hoc variant were the security level of the key and MACs should be chosen according to the expected delays. For safety reasons, in the case of lower priority for the broadcast protocol, one can also enforce a key-recovery procedure. As re-initialization of the protocol will be too expensive, for this purpose one can use safety-chains, i.e., another upper layer of key-chains that are computed a-priori by the sender to allow recovery after an error has occurred. This allows a trade-off between traffic priority and computational power.

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. Our test setup consists of a master node and multiple slave nodes. The communication master holds the key chains which are used to send authenticated messages to slave nodes.

A. Employed platforms

We made tests on Freescale S12X and Infineon TriCore, two different classes of microcontrollers used in automotive and industrial applications. The controller specifications are described next.

1) The S12X chip: We used a SofTec Microsystems ZK-S12-B development board shipped with MC9S12XDT512, a Freescale 16-bit automotive microcontroller from the S12X family. All members of this family are equipped with a co-processor called XGATE which can be used to increase the microcontroller's data throughput. We made use of this module to increase the efficiency of our implementation by assigning it the task of computing cryptographic functions. The S12X microcontroller that we used has 512kBytes of flash memory and 20kBytes of RAM. Both FLASH and RAM memories are banked. Thus, a total of 8kBytes of RAM space can be used for continuous allocation while the rest of the RAM can be accessed in a 4kByte window. The data-sheet specifies that the maximum bus frequency that can be set using the PLL module is 40MHz. We configured the PLL for frequencies beyond this specified value and were able to go up to 80MHz without introducing any abnormal behavior. The on-chip CAN module can be programmed to function at bit-rates up to 1 Mbps. The limitation for the maximum achievable CAN speed on the S12 development board is given by the on-board low speed fault tolerant transceiver which can only run at speeds up to 125 kbps.

2) The TriCore chip: We also used Infineon TriCore microcontrollers from the AUDO (AUtomotive unifieD processOr) family. All members of this family are 32 bit microcontrollers built for computational performance. For the communication master we used a TC1797 microcontroller which fits in the high-end segment. The TC1797 from the AUDO Future family can work at frequencies up to 180MHz, has a program flash memory of 4MB and a 64kB data flash along with 176kB of SRAM. The slave node was implemented on a TC1782 which is a mid range automotive microcontroller and a member of the newer AUDO MAX family (but it is still based on the same 1.3.1 core version as TC1797). TC1782 has the same operating frequency and SRAM size as TC1797 while the program flash memory is reduced to 2,5MB and the data flash is double in size to hold 128kB.

B. Computational performance

First, we evaluated the performance of the microcontrollers for computing cryptographic primitives. Table II holds the results of our measurements (performance cuts off at multiples of 55 bytes due to block size and padding, input sizes are chosen based on some test vectors in RFC-1321). 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. The two TriCore microcontrollers have the same core version thus identical performances. 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).

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 the XGATE side, 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

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

TABLE III
S12X AND TRICORE ROUND TRIP TIME.

Nonce size (bytes)	MAC size (bytes)							
	1	2	4	8	1	2	4	8
	S12X (ms)				TriCore (μs)			
2	3.740	3.810	3.935	4.185	172.4	182.4	205.5	241.5
4	3.870	3.935	4.055	4.295	189.4	199.5	219.5	258.5
8	4.160	4.220	4.355	4.620	225.5	236.5	256.5	295.5

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 $1.73ms$) which could be considered too slow in a time critical system.

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. Table III holds or measurements for different nonce and MAC sizes. For these measurements, the CAN bus speed was set to 1Mbps and HMAC-MD5 with an 8 byte key was used for computing the MAC. The length of the computed MAC is 16 bytes and we obtained the smaller MACs by truncating this original value to the desired length.

As Table III show there is little difference in round-trip time if we change the nonce or MAC size from 1 to 8 bytes, since only one CAN frame is sent in both cases. When we change these sizes to 16 bytes the increase in round-trip time is greater due to the overhead caused by having to send two CAN frames instead of only one as the case for 1 and 8 bytes messages.

C. Adjusting parameters

Choosing the best combination of parameters highly depends on the application and on the devices used for implementation. We therefore tried different values for the protocol parameters on the TriCore implementation to find the best suited setup. The parameters were chosen so that they fit for authenticated communication over a time span of 10 years

TABLE IV
SOME PARAMETERS CHOICES FOR THE INFINEON TRICORE PLATFORM.

Key size	3 levels			3 levels		
(bytes)	$\delta(ms)$	M (bytes)	σ	$\delta(ms)$	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)	$\delta(\mu s)$	M (bytes)	σ	$\delta(\mu s)$	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

without the need of reinitialization. Table IV 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 $10ms$ 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 $471.44\mu s$ 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 $315.24\mu s$ for a 3 level setup and $340.8\mu s$ 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 $343.64\mu s$ while the 8 byte keys on the second level will have a life time of $84.88ms$. 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 $10ms$ 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.

References

[1] L. Almeida, P. Pedreiras, and J.A.G. Fonseca. The FTT-CAN protocol: why and how. *Industrial Electronics, IEEE Transactions on*, vol.49, no.6, pp. 1189- 1201, 2002.

[2] R. Anderson, F. Bergadano, B. Crispo, J.-H. Lee, C. Manifavas, and R. Needham. A new family of authentication protocols. *SIGOPS Oper. Syst. Rev.*, 32:9–20, October 1998.

[3] H. K. Aslan. A hybrid scheme for multicast authentication over lossy networks. *Computers & Security*, 23(8):705 – 713, 2004.

[4] H. Bar-El. Intra-vehicle information security framework. In *Proceedings of 9th Embedded Security in Cars Conference, ESCAR*, September 2009.

[5] M. Barranco, J. Proenza, and L. Almeida. Quantitative comparison of the error-containment capabilities of a bus and a star topology in CAN networks. *Industrial Electronics, IEEE Transactions on*, 99, 2009.

[6] M. Barranco, J. Proenza, G. Rodriguez-Navas, and L. Almeida. An active star topology for improving fault confinement in CAN networks. *Industrial Electronics, IEEE Transactions on*, 2, 2, 78–85, 2006.

[7] D. Berbecaru, L. Albertalli, and A. Lioy. The ForwardDiffSig scheme for multicast authentication. *IEEE/ACM Transactions on Networking*, 18:1855–1868, December 2010.

[8] F. Bergadano, D. Cavagnino, and B. Crispo. Individual authentication in multiparty communications. *Computers & Security*, 21(8):719–735, 2002.

[9] G. Cena, and A. Valenzano. An improved CAN fieldbus for industrial applications. *Industrial Electronics, IEEE Transactions on*, vol.44, no.4, 553–564, 1997.

[10] G. Cena, and A. Valenzano. FastCAN: a high-performance enhanced CAN-like network. *Industrial Electronics, IEEE Transactions on*, vol.47, no.4, pp.951-963, 2000.

[11] Y. Challal, A. Bouabdallah, and H. Bettahar. H2A: Hybrid hash-chaining scheme for adaptive multicast source authentication of media-streaming. *Computers & Security*, 24(1):57 – 68, 2005.

[12] J. Charzinski. Performance of the error detection mechanisms in CAN. In *Proceedings of the 1st International CAN Conference*, pages 20–29, 1994.

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

[14] M. Cheminod, A. Pironti, R. Sisto, Formal Vulnerability Analysis of a Security System for Remote Fieldbus Access, *Industrial Informatics, IEEE Transactions on* , vol.7, no.1, pp.30-40, Feb. 2011.

[15] D. Dzung, M. Naedele, T. P. Von Hoff, and M. Crevatin. Security for Industrial Communication Systems. *Proceedings of the IEEE*, 93(6):1152–1177, Feb. 2005.

[16] N. Falliere, L. O. Murchu, and E. Chien. W32.Stuxnet dossier. Technical report, Symantec, 2011.

[17] W. Granzer, F. Praus, W. Kastner. Security in Building Automation Systems *Industrial Electronics, IEEE Transactions on* , vol.57, no.11, pp.3622-3630, Nov. 2010.

[18] B. Groza and P.-S. Murvay. Higher layer authentication for broadcast in controller area networks. In *International Conference on Security and Cryptography (SECRYPT)*, 2011.

[19] H. Hartenstein and K. Laberteaux. *VANET Vehicular Applications and Inter-Networking Technologies*. Wiley, 2009.

[20] 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 *Security and Privacy (SP), 2010 IEEE Symposium on*, pages 447 –462, May 2010.

[21] L. Lamport. Password authentication with insecure communication. *Commun. ACM*, 24:770–772, November 1981.

[22] K. Lemke, C. Paar, and M. Wolf. *Embedded Security in Cars Securing Current and Future Automotive IT Applications*. Springer Verlag, 2006.

[23] D. Liu and P. Ning. Efficient distribution of key chain commitments for broadcast authentication in distributed sensor networks. In *Proc. of the 10th Annual Network and Distributed System Security Symposium*, pages 263–276, 2003.

[24] D. Liu and P. Ning. Multilevel μtesla: Broadcast authentication for distributed sensor networks. *ACM Transactions on Embed. Comput. Syst.*, 3:800–836, November 2004.

[25] P. Marti, A. Camacho, M. Velasco, M. El Mongi Ben Gaid, Runtime Allocation of Optional Control Jobs to a Set of CAN-Based Networked Control Systems, *Industrial Informatics, IEEE Transactions on*, vol.6, no.4, pp.503-520, Nov. 2010

[26] R. C. Merkle. A digital signature based on a conventional encryption function. In *A Conference on the Theory and Applications of Cryptographic Techniques on Advances in Cryptology*, CRYPTO '87, pages 369–378, London, UK, 1988. Springer-Verlag.

[27] R. Mitchell. *Tutorial: Introducing the XGATE Module to Consumer and Industrial Application Developers, March 2006.* Freescale, 2004.

[28] 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. In *Industrial Electronics, IEEE Transactions on*, vol.58, no.7, pp.3090-3100, Jul. 2011.

[29] A. Perrig, R. Canetti, D. Song, and J. D. Tygar. Efficient and secure source authentication for multicast. In *Network and Distributed System Security Symposium, NDSS '01*, pages 35–46, 2001.

[30] A. Perrig, R. Canetti, D. Song, and J. D. Tygar. Spins: Security protocols for sensor networks. In *Seventh Annual ACM International Conference on Mobile Computing and Networks (MobiCom 2001)*, pages 189–199, 2001.

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

[32] Robert BOSCH GmbH. *CAN Specification Version 2.0.*, 1991.

[33] ECRYPT II, *Yearly Report on Algorithms and Keysizes (2009-2010)*, Revision 1.0, 2011.

[34] International Organization for Standardization. *ISO 11898-1. Road vehicles - Controller area network (CAN) - Part 1: Controller area network data link layer and medium access control*, 2003.

[35] International Organization for Standardization. *ISO 11898-4. Road vehicles - Controller area network (CAN) - Part 4: Time triggered communication*, 2004.