Broadcast Authentication in a Low Speed Controller Area Network [⋆]

Bogdan Groza and Pal-Stefan Murvay

Department of Automatics and Applied Informatics,
Politehnica University of Timisoara, Romania
`bogdan.groza@aut.upt.ro, stefan.murvay@gmail.com`

Abstract. Controller Area Network (CAN) is a communication bus that has no cryptographic protection against malicious adversaries. Once isolated, the environments in which CAN operates are now opened to intruders and assuring broadcast authentication becomes a concern. To achieve this, public key primitives are not a solution because of the computational constraints, but symmetric primitives can be used with time synchronization at the cost of additional delays. Here we study several trade-offs on computational speed, memory and bandwidth having the main intention to depict the lower bounds on the efficiency of such protocols. For this purpose we use a wide spread controller from Freescale located somewhat on the edge of the market capable of low speed, fault tolerant CAN communication. To further improve the computations we also make use of the XGATE co-processor available on the S12X derivative. The performance of both hash functions and block ciphers is examined for efficient construction of the key chains.

Keywords: authentication; broadcast; controller area network.

1 Introduction and related work

Controller Area Network or simply CAN is a communication bus initially developed by BOSCH to be used by controller units in vehicular systems [5]. The initial specifications are now superseded by ISO 11898 [6] while its area of application also extended outside vehicles to automation applications in general. Although high performance buses were developed in the last decade, such as FlexRay, because of its efficiency and reduced cost CAN is still the most commonly used communication bus in automotives today.

Traditionally, in control systems in general and in automotives in particular, reliability was a main concern but only with respect to natural phenomenons (electromagnetic disturbances, thermal noise, etc.) or accidents of various causes but not in front of Dolev-Yao adversaries. Thus CAN has very efficient mechanisms to deal with errors and to recover afterwards. In fact, 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. For the interested

[⋆] This is a revised postproceedings version of the paper entitled *Higher Layer Authentication For Broadcast In Controller Area Networks* presented at *SECRYPT'11* reader, an in-depth study of the performance of CAN error detection mechanism was done by Charzinski in [3].

However, in the last decade, industrial control systems and automotives become opened to malicious adversaries as well and a significant part of the security community focused on this kind of issues. A recent comprehensive book for security and in particular cryptographic security in automotives is [10] but a high amount of papers were published since then.

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 opened for intruders. Recent research showed current automobiles to be unexpectedly vulnerable to external adversaries [8] and it is likely that many other environments in which CAN operates are not completely isolated from the outside world. Security in front of such adversaries can be achieved by implementing this at the application level. In fact such improvements happened in the past, for example when deterministic delays were needed on the CAN bus with the development of Time Triggered CAN [7]. Still, to best of our knowledge there is no implementation available to assure authenticity in CAN networks. Thus, the main intention of this paper is to develop a higher layer implementation and to study several trade-offs to increase its efficiency. We analyze this both at a theoretical level by introducing the corresponding formalism and by designing an efficient protocol and at a practical level by following an efficient implementation. This is done on S12X microcontrollers from Freescale, a family of microcontrollers commonly used in the automotive industry, with the use of the XGATE co-processor available on S12X derivatives to speed up cryptographic functions.

As for the cryptographic mechanism that can be employed for this purpose, public-key cryptography is not the solution 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 will require thousands of bits and cause a significant overhead. Elliptic curves will significantly reduce the size of the messages, 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 largely increase the cost of components, an issue that is largely avoided by manufacturers.

In contrast, symmetric primitives were efficiently employed for authentication in constrained environments such as sensor networks [15], [11], [12]. Due to the broadcast nature of CAN, protocols similar in nature to the well known TESLA protocol [16], [14] can be used in this context as well. 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 [9]. The work of Bergadano et al. [2] 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 [1]. 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 [11], [12] and several variants of the protocols are presented by Perrig as well in [16], [14] .

In the case of the industrial controllers, 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. But while low computational power gives some similarities, other constraints are different. For example, energy consumption is a relevant issue in sensor networks, but usually for control units inside a car this is not a main concern since they do not strongly rely on small batteries. On the other side, a different constraint here, that is not so prevalent in sensor networks, is the size of the message which is limited to 64 bits on a CAN frame. Indeed, larger messages can be split in smaller messages but the overhead inflicted by the structure of the CAN frame is around 50%. This becomes prevalent in the case of one-way chain based protocols, where hash-functions are used to compute the chain elements and thus to send an element of the chain will require at least two exchanged messages (assuming the simplest hash function outputs 128 bits). To this, one will need to add the message authentication code as well, which again requires at least two exchanged messages, etc. Thus, at least four CAN frames are needed to transmit just the security elements of one frame of useful information. Still, the most critical part, in automotive communication and control systems in general, where this protocol is mostly used, are the authentication delays, i.e., how fast a packet can be deemed as authentic. For this purpose, the most relevant constraint to which we want to give a positive answer is the authentication delay. 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 on the bandwidth. Using too large chains means too much time in the initialization stage and large amounts of memory, while too short chains means either high authentication delays or too frequent re-initializations, etc. Depicting an optimum in this context is not straight forward and we study this in detail in what follows. In particular, we used in our scheme several levels of one way chain. While three levels of one-way chains were reported to be closed to optimal in sensor networks, due to memory constraints and to reduce initialization in some situations we used more levels. This is because of both the time horizon of the protocol and of the duration of the disclosure interval. In sensor networks the disclosure interval was in the order of tens or hundreds of milliseconds, while here, to increase communication speed we want to reduce this as much as possible. Of course, we are finally limited by the bus speed at 128 kbps and by the synchronization error, which in fault-tolerant CAN will not allow us to drop the disclosure delay under several milliseconds. Practical examples are given in the experimental results section.

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 experimental results concerning the implementation of the proposed protocol on S12X microcontrollers. This includes experimental results for the implementation of cryptographic primitives on the XGATE co-processor. Section 4 holds the conclusions of our paper.

2 The protocol

From an upper view the design paradigm is the following. Memory, computational speed and bandwidth give bounds on the length of the chains that can be used and the number of levels. This, along with the synchronization error, further bounds the authentication delay, as messages cannot be authenticated faster than the disclosure delays. Further, to improve on the delays, we allocate equidistant timings in order to avoid a non-uniform load on the CAN bus. Indeed, since initialization packets must have the highest priority, if timings are non-uniform, there will be periods when more chains need to be initialized and thus the bus will be heavily loaded by initialization packets. Before getting to the description of the protocol we briefly enumerate in what follows some relevant aspects of the CAN bus.

2.1 Short description of the CAN protocol

We are not interested here in typical aspects of the protocol such as error detection, synchronization, etc., so we will not mention them. CAN bus has a broadcast nature. Nodes are connected by a two wire bus topology, as shown in Figure 1, and access to the bus is gained with priority based on a message identifier which forms the first part of a frame. This identifier has 29 bits in extended frames and 11 bits in standard frames. The structure of the CAN frame consists in the arbitration field (the identifier), 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. Few words on arbitration are in order. The way of arbitrating is by judging the winner 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. 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 128kbps and high-speed nodes that work up to 1Mbps.

2.2 Overview of the authentication protocol

We use time synchronization and multiple levels of one-way chains in order to authenticate the broadcast from a particular node. The generic structure of the key chains is depicted in Figure 2. Packets arriving on the communication bus are depicted as well,

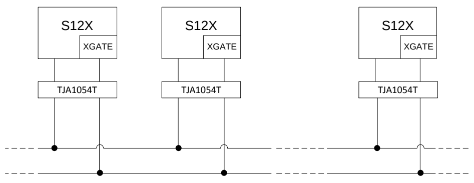

Fig. 1: Fault tolerant CAN bus topology with S12X controllers and TJA1054 transceivers

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. Packet P_i is sent to initialize a chain from level i. In what follows we will use the following notations:

– L - number of chain levels,
– $\lambda_i, i = 0..L$ - length of the chain on level i,
– $\delta_i, i = 0..L$ - disclosure delay on level i,
– B - bus speed, normalized to packets per second (one packet can carry one key),
– M - available memory (measured in elements of the key chain that can be stored),
– S - computational speed (number of keys that can be computed per second),
– T - time horizon of the protocol,
– t - time as integer value, a subscript indicates particular details.

Based on these notations in the next section we discuss the optimal allocation of the broadcast parameters.

In principle we need two distinct protocols: an initialization protocol and a broadcast protocol. The role of the initialization protocol is to allow each unit to commit its initialization values and to achieve time synchronization with other participants. This part of the protocol can rely on more expensive operations required by public-key cryptography. In this stage we consider that each principal will authenticate itself by using a public key certificate that is signed by a trusted authority. Initial authentication based on public-key infrastructure is important to assure composability. This ensures that different components, from potentially different manufacturers, can be assembled in one system and is a common demand of the market today. For example, in a different context (that of communication alone), the latest state-of-the-art protocol, FlexRay, has communication segments that are preallocated such that different components from different providers can be bind into a system. Thus we require that each node must store the public key of a trusted authority. The initialization protocol must also ensure time synchronization. This 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. However, in our scenario a digital signature costs tens, or hundreds milliseconds, which will result in a high tolerance margin that will further require 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 magnitudes faster, in particular in our experiments the round-trip-time was less than 2 milliseconds. 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 $[t_B + t_1 - t_0,\ t_B + t_1 - t_0 + \varepsilon_{AB}]$. Further, the initialization values for the chains can be shared between each node and the central node who can broadcast them to all communication participants. We will not insist on details of the initialization protocol which is done in the initial phase with no constraints.

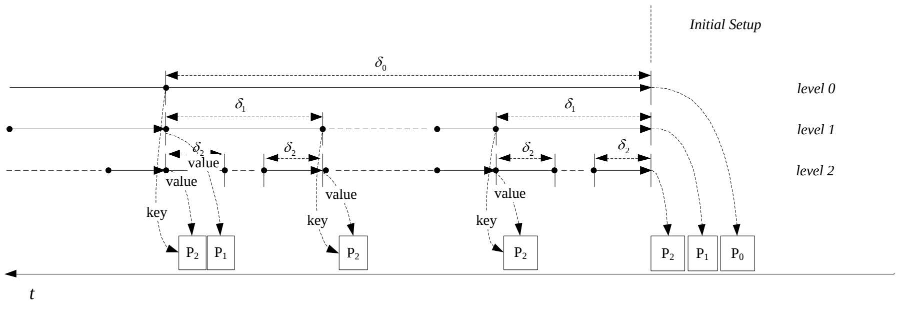

Fig. 2: Basic structure of the key chains and packets as they are dropped on the bus

2.3 Optimal allocation of key chain lengths and levels

For brevity we consider a homogeneous network with nodes that have equal computational abilities, thus the same computational delays and lengths for the chains can be handled by all of them. Otherwise, the protocol can be scaled according to the computational abilities of each client, but we want to keep the model as simple as possible.

To formalize the optimal allocation of chain lengths and levels, different to previous work, we use a tolerance relation to define the lengths of the chains and the disclosure delays at each level. The tolerance relation is formed by vectors which are defined as initialization values for each communication participant.

Definition 1. We say that a set of pairs $< \lambda_i, \delta_i >$ forms tolerance relation with respect to memory, computational speed, bandwidth and time, denoted as $< \Lambda, \Delta >_{M,S,B,T}$ if the following constraints hold:

$$\sum_{i=0,L} \lambda_i = \lambda_0 + \lambda_1 + ... + \lambda_L \leq M \tag{1}$$

$$\lambda_0 + (\lambda_0+1)\cdot\lambda_1 + (\lambda_0+1)\cdot(\lambda_1+1)\cdot\lambda_2+... \\ +\ (\lambda_0+1)...(\lambda_{L-1}+1)\lambda_L = \sum_{i=0,L}\lambda_i\prod_{j=0}^{i-1}(\lambda_j+1) = \prod_{i=0}^{L}(\lambda_i+1)-1 \le B\cdot T \quad (2)$$

$$\lambda_0 + (\lambda_0+1)\cdot\lambda_1 + (\lambda_0+1)\cdot(\lambda_1+1)\cdot\lambda_2+... \\ +\ (\lambda_0+1)...(\lambda_{L-1}+1)\lambda_L - \lambda_0 - ... - \lambda_L = \prod_{i=0}^{L}(\lambda_i+1)-1-\sum_{i=0}^{L}\lambda_i \le S\cdot T \quad (3)$$

Relation (1) gives the memory bound, i.e., the sum of the lengths of the chains cannot exceed the total memory. Relation (2) and (3) are bounds on bandwidth and computational time, i.e., the total number of elements of the one-way chain cannot exceed the available bandwidth for transmission and cannot require more computational time than available over protocol lifetime T. Values $\lambda_0,\lambda_1,...,\lambda_L$ are subtracted from relation (2) to get (3) since the first key chain on each level is computed in the initialization stage. Indeed, relations (2) and (3) can be further refined for the disclosure delays on all up to the last level since the re-initialization of each chain should be done in the disclosure delay of the previous level. We introduced this definition to keep our presentation formal, but it is obvious that defining good tolerance relations is a matter of good engineering.

Remark 1. Relations (2), (3) correspond to the case when all chains are committed in the initialization stage and each element from each level commits a new chain from all levels below. This can be modified to the case when only the keys on the first level are committed in the initialization stage and further each element on each level commits a new chain only on the first level below. This will give cleaner tolerance relations: $\sum_{i=1,L}\prod_{j=0,i}\lambda_j = \lambda_0\cdot\lambda_1+...+\lambda_0\cdot\lambda_1\cdot...\cdot\lambda_L \le S\cdot T$ and $\sum_{i=0,L}\prod_{j=0,i}\lambda_j = \lambda_0+\lambda_0\cdot\lambda_1+...+\lambda_0\cdot\lambda_1\cdot...\cdot\lambda_L \le B\cdot T$. While in this way the number of commitments will be constant 1, the lifetime of the protocol will be reduced as fewer chains are committed. Otherwise, the number of commitments increases, but at most to the number of levels (usually 3 or 4) which should be easily supported on the bus. This appears to be preferable since it improves the protocol lifetime.

Remark 2. The general relation between chain lengths and disclosure delays will now be the following: $\delta_i = \delta_{i-1}/(\lambda_i+1), i>0$. Intuitively, this also means that it must be feasible to compute and send λ_i chain elements in time δ_{i-1}. Thus λ_i can also be defined as a function of δ_{i-1} considering the computational power of the device.

Remark 3. Given a tolerance relation $<\Lambda,\Delta>_{M,S,B,T}$ with respect to time, space and bandwidth the disclosure delay and the computational overhead have an inverse variation. Thus: the minimal disclosure delay is achieved if chains are of equal size, i.e., $\lambda_i = M/L$, while the minimal computational and communication overhead is achieved if upper level chains are smaller, i.e., $\lambda_0 < \lambda_1 < ... < \lambda_L$. This is intuitively since the product of two values whose sum is fixed is maximal if the two values are equal and minimal if one of the values is 1. For example, assume $x+y=z$ then $\forall k \ge 1, z/2\cdot z/2 > (z/2-k)\cdot(z/2+k) = z^2/4-k^2$. Now, to achieve the minimum delay, the product of the chain lengths $\lambda_0 \cdot \lambda_1 \cdot ... \cdot \lambda_L$ must be maximal. If we split this product exactly into the half left and half right terms, assuming an even number of terms which is wlog, then the maximum product is achieved if: the left and right terms are maximal and equal, and so on. To achieve minimal bandwidth and computational requirements we need $\lambda_0 \cdot \lambda_1 + ... + \lambda_0 \cdot \lambda_1 \cdot ... \cdot \lambda_L$ to be minimal. This can be written as $\lambda_0 \cdot (\lambda_1 + ... + \lambda_1 \cdot ... \cdot \lambda_L)$ and as right term cannot be equal to 1 the left term must be in order to minimize the product and so on. Thus, the optimum choice of lengths with respect to delays, is to have all chains of equal size. For the purpose of generality, as well as for the fact that in practice small variations between chain lengths may occur when the amount of available memory is not directly divisible with the number of levels, we will keep the following exposition for the case when chains are of arbitrary sizes.

2.4 Optimal allocation of timings

By optimal allocation of the key disclosure time we want to achieve minimal delays for sending messages and authenticating them. Of course, the authentication delay is bounded by the disclosure delay, i.e., packets cannot be authenticated sooner than the disclosure delay from the last level. This bound was already fixed by the tolerance relation. However, the straight forward mechanism suggested in Figure 2, in which chains are re-initialized successively, causes more overhead at the disclosure time of keys from upper layer chains (since at this time all chains from lower levels need to be re-initialized as well). To overcome this, we define a procedure which we call *equidistant timing* by which all packets are disclosed at periods of time separated by equal delays. More, we will use chains from upper levels to authenticate information packets as well and not only forthcoming key chains. Thus, we will normalize the disclosure time to:

$$\delta_{norm} = \frac{T}{\prod_{i=0}^{L}(\lambda_i + 1) - 1} \tag{4}$$

This relation comes from the assumption that each tip of a chain is committed by a MAC code but once it is not released until the precise time when it can be already used for authentication. If the tip of the chain is also sent with the MAC code, then the denominator will be $\prod_{i=0}^{L} \lambda_i - 1$. In the forthcoming scheme, keys on all levels are released at δ_{norm} intervals. Of course, if we use relation (4), δ_{norm} is equal to δ_L but we prefer to use this notation to avoid confusion with a generic scheme in which this may not happen, i.e., not all keys are released at δ_{norm} but only the keys from the chain on level L.

In what follows we need to establish three things: i) which is the disclosure time for a particular key k (Definition 4), ii) given a particular time t which key must be disclosed, or which is the last key that was disclosed upon t (Remark 4) and iii) given a particular packet, containing authentication information, what condition must be met on the receiver's side to deem this packet as on-time (Definition 5).

To determine all these, the easiest way to decide is by writing the time with respect to the vectors of the tolerance relation which is established by the next definition.

Definition 2. Given a discrete time value t by $t_{<\Lambda,\Delta>} =< t_0 ... t_L >$ we denote the decomposition of t with respect to the lengths of the chains $(\lambda_0, \lambda_1, ..., \lambda_L)$ and normalized time (δ_{norm}) as a basis. In this way, each element from $t_{<\Lambda,\Delta>}$ gives the last element that was released on the corresponding level. That is, given $t' = t \bmod \delta_{norm}$ we write:

$$\begin{aligned} t &= t_0 \cdot (\lambda_1 + 1) \cdot \ldots \cdot (\lambda_L + 1) \cdot \delta_{norm} + \ldots + t_{L-1} \cdot (\lambda_L + 1) \cdot \delta_{norm} + t_0 \cdot \delta_{norm} + t' \\ &= \delta_{norm} \cdot \sum_{i=0..L} t_i \cdot \prod_{j=i+1..L} (\lambda_j + 1) + t' \end{aligned} \tag{5}$$

Sender's perspective. For the moment we consider the case of a single sender. Let t_{start} denote the time at which the broadcast was started and assume that it is started by the communication master which is also responsible for time synchronization. Thus t_{start} is the exact time at which the broadcast started (no drifts for the sender).

Definition 3. Let $\tau_{\text{left}} = \tau_0\tau_1...\tau_{i-1}$ denote a vector of i positive integers such that each element on position i is less or equal to λ_i. Given a tolerance relation $<\Lambda,\Delta>_{M,S,B,T}$, an indexed key K_τ is a key entailed by a vector τ. An indexed key chain $\mathcal{K}_{<\Lambda,\Delta>}$ is a collection of indexed keys, derived as follows: having a fixed fresh seed K_{master}, a key derivation process $\mathcal{KD}$ and a one-way function $\mathcal{F}$, then $\forall i \in [0,L], \tau_i \in [1,\lambda_i]$, given $K_{\tau_{\text{left}}|0|\bar{0}} = \mathcal{KD}(K_{\text{master}}, \tau_{\text{left}})$:

$$K_{\tau_{\text{left}}|\tau_i|\bar{0}} = \mathcal{F}(K_{\tau_{\text{left}}|\tau_i+1|\bar{0}}) \tag{6}$$

Definition 4. Let $\mathcal{DT}(K_\tau)$ denote the disclosure time of the indexed key K_τ. Given a broadcast started at t_{start} the disclosure time of this key is:

$$\mathcal{DT}(K_\tau) = t_{start} + \delta_{norm} \cdot \sum_{i=0..L} \tau_i \cdot \prod_{j=i+1..L} (\lambda_j + 1) \tag{7}$$

Definition 4 allows the creation of chains with the structure from Figure 2 while definition 5 allows keys to be released at equal time intervals δ_{norm}.

Remark 4. The key released by the sender at time $t_{current}$ given a broadcast started at t_{start} is $K_{t_{<\Lambda,\Delta>}}$ where $t = t_{current} - t_{start}$. This means that the key is from level l, where l is the first non-zero index from right to left and the position of the key on the key chain from level l is t_l.

Reinitialization packets and efficiency. To avoid a non-uniform bus load, as discussed previously, re-initialization packets will be dropped *equidistantly* in the δ_{norm} intervals. Otherwise, packets carrying data must be delayed until all re-initialization packets are sent. This is because re-initialization packets must have priority on the bus, otherwise the protocol will succumb an will require a new initialization stage which is even more expensive. Thus we can also use as an efficiency criteria the maximum delay until a new CAN frame can be send. For the basic scheme the maximum delay fluctuates with the number of initialization packets. This delay can be easily established for the basic scheme. Let $z_i(x)$ denote the number of consecutive zeros in vector x starting from the rightmost position. At discrete time value t, given $t_{<\Lambda,\Delta>}$ the delay until the next packet can be sent is:

$$delay = \max[0, z((t - t_{start})_{<\Lambda,\Delta>})] \tag{8}$$

Indeed, the number of consecutive zeros at the end of the time value denotes how many chains need to be initialized at that particular time. This value becomes constant for the equidistant scheme and data packets are delayed by at most one packet.

To complete the view on efficiency, we should also define the overhead induced by the authentication mechanism. The overhead has two distinct components, the *authentication overhead* which is the overhead inflicted by the authentication keys that are released on the bus, and the *re-initialization overhead* which is the overhead inflicted by the re-initialization material, i.e., MAC codes that commit the key chains. Indeed, to this one may want to add the overhead induced by the message authentication codes, MACs, associated to each data packet that is send over the bus. We will not take this into account because this is application dependent, not protocol dependent, indeed 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. In other applications it may be the reverse, and data packets can be large while the MAC will not significantly increase the overhead. Based on these we can also define the total overhead inflicted by the protocol.

Remark 5. Given a tolerance relation $< \Lambda, \Delta >_{M,S,B,T}$ the authentication and re-initialization overheads and the bus load induced by the protocol is: $OH_{auth} = B^{-1} \cdot \prod_{i=0}^{L}(\lambda_i + 1) - 1$ (as given in relation (2), the right term of the product covers all key released on the bus), $OH_{reinit} = B^{-1} \cdot \sum_{i=1}^{L}(\prod_{j=0}^{i-1}(\lambda_j + 1) - 1)$ (the right term of the product gives al the key chains that are committed on the bus) and $OH_{total} = OH_{auth} + OH_{reinit}$.

One may note that if the authentication delay is bigger the overhead also becomes lower since fewer elements of the chain are sent. Also, the re-initialization overhead increases with the number of levels. We give concrete examples for these in the experimental results.

Figure 3 shows the influence of chain length on bus overhead and disclosure delays. Plots are given for three and four levels of key chains. We note that the delays drop rapidly by increasing the number of levels, but in the same manner the overhead increases (at 100% the bus is locked and communication halted). Figure 3 shows the variation of memory requirements with the number of chain levels, which is the same as the initialization time. All plots are taken for a time range of 24 hours, the delay is fixed to 5 ms.

Receiver's perspective. We consider the case of a sender $\mathcal{S}$ with synchronization error $\varepsilon_{\mathcal{S}}$ and a receiver $\mathcal{R}$ with synchronization error $\varepsilon_{\mathcal{R}}$. 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 errors $\varepsilon_{\mathcal{S}}$ and $\varepsilon_{\mathcal{R}}$ for sender and receiver, an authentication packet indexed by τ must discarded unless:

$$\mathcal{T}_{\text{rec}}(P) \leq \mathcal{DT}(\tau) + \varepsilon_{\mathcal{S}} + \varepsilon_{\mathcal{R}} \tag{9}$$

Here $\mathcal{T}_{\text{rec}}(P)$ denotes the estimated time on the sender's side computed by the receiver when packet P arrives. If the sender is also responsible for time synchronization then $\varepsilon_{\mathcal{S}} = 0$.

Protocol description. We can now summarize previous notions in one definition for the entire protocol.

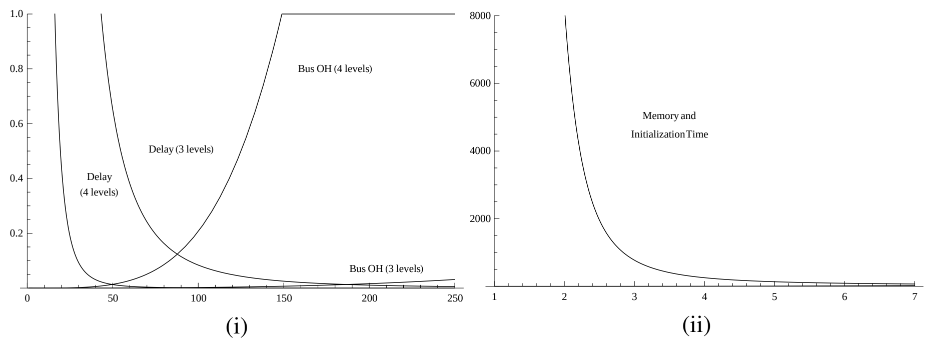

Fig. 3: Delay and overhead variation with chain length (i) and memory, initialization time and re-initialization packets variation with number of levels (ii)

Definition 6. Given tolerance relation $< \Lambda, \Delta >_{M,S,B,T}$, an indexed key chain $\mathcal{K}_{<\Lambda,\Delta>}$ and the two roles sender and receiver denoted by $\mathcal{S}$, $\mathcal{R}$ each with synchronization errors $\varepsilon_{\mathcal{S}}$, $\varepsilon_{\mathcal{R}}$ with respect to a common clock, protocol Broadcast$[\mathcal{S}, \mathcal{R}, < \Lambda, \Delta >_{M,S,B,T}, \mathcal{K}_{<\Lambda,\Delta>}]$ is formed by the following two rules for the two roles:

1. $\mathcal{S}$ sends K_τ at $\mathcal{DT}(K_\tau)$ and the message M and its corresponding $MAC(K_\tau, M)$ in any empty time-slot with the condition that $MAC(K_\tau, M)$ is released no latter than $\mathcal{DT}(K_\tau) + \xi$,
2. $\mathcal{R}$ discards all $MAC(K_\tau, M)$ received later than $\mathcal{DT}(K_\tau) + \varepsilon_{\mathcal{S}} + \varepsilon_{\mathcal{R}}$ and deems authentic all other messages for which $MAC(K_\tau, M)$ can be verified with an authentic key. A key $K_{\tau_{\text{left}}|\tau_i|\bar{0}}$ is authentic only if $K_{\tau_{\text{left}}|\tau_i|\bar{0}} = \mathcal{F}(K_{\tau_{\text{left}}|\tau_i+1|\bar{0}})$ and $K_{\tau_{\text{left}}|\tau_i|\bar{0}}$ is a previously received/computed authentic key.

Here ξ denotes a tolerance margin until the sender can send a MAC with a particular key. 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 parameter for the protocol and it must hold that $\varepsilon_{\mathcal{S}} + \varepsilon_{\mathcal{R}} << \xi$. 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.

Broadcast$[\mathcal{S}, \mathcal{R}, < \Lambda, \Delta >_{M,S,B,T}, \mathcal{K}_{<\Lambda,\Delta>}]$ is a secure broadcast authentication protocol. The security of this family of protocols is well established, the informal argument is that the adversary cannot construct $MAC(K_i, M)$, until K_i is released. As function $\mathcal{F}$ is one-way he cannot derive K_i from $\mathcal{F}(K_i)$ and by the time K_i is released any MAC with K_i that is further received will be discarded. Formal proofs for such protocols can be found in [14], [2].

2.5 The case of multiple senders

The case of k senders can now be easily derived from the previous formalism. To preserve the *equidistant* time schedule we use the nominal disclosure time δ_{norm} and divide it by the number of senders k. Let us define the next sender delay as:

$$\delta_{next} = \frac{\delta_{norm}}{k} \tag{10}$$

Definition 7. Let $\mathcal{DT}(K_\tau)$ denote the disclosure time of an indexed key by the by the k^{th} sender. Given a broadcast started at t_{start} we have:

$$\mathcal{DT}(K_\tau, k) = \mathcal{DT}(K_\tau) + k \cdot \delta_{next} \tag{11}$$

The security conditions which has to be verified by receivers must also add the $k \cdot \delta_{next}$ term to the disclosure time of the k-th sender.

3 Application setting and experimental results

As stated, for the implementation of the previously described protocol, we used a Freescale 16-bit automotive grade microcontroller (MC9S12XDT512) from the S12X family on SofTec Microsystems ZK-S12-B development board. One special feature of this family is the presence of an incorporated co-processor called XGATE which can be used to increase the microcontroller's data throughput [4]. We made use of this module to increase the efficiency of our implementation by assigning it the task of computing the underlying cryptographic functions.

The S12X microcontrollers used in our experiments have 512kbytes of FLASH memory and 20kbytes of RAM. Both FLASH and RAM memories are banked as a consequence of the 16 bits wide address bus which is not sufficient to access all memory locations. Thus, a total of 8kbytes of RAM space can be used for continuous allocation while the rest of the RAM can be accessed in 4kbyte windows. The amount of RAM memory that can be used for storing key chains is relevant as it determines the maximum number of chain levels and their lengths.

The maximum bus frequency that can be set using the PLL module is, according to the data-sheet, 40MHz. We configured the PLL for frequencies beyond this specified value and were able to go up to speeds of 80MHz. After assessing that the behaviour of the microcontroller at this overclocked frequency is normal we used it in our tests and compared the results with the ones obtained for 40MHz.

The on-chip CAN module implements version 2.0A/B of the CAN protocol and supports a programmable bit-rate up to 1 Mbps. A limitation for the maximum achievable CAN speed comes from the on board low speed fault tolerant transceiver which can only run at speeds up to 125kbaud.

3.1 XGATE module

The XGATE module has a built in RISC core with instructions for data transfers, bit manipulations and basic arithmetic operations. The RISC core can access the internal memories and peripherals of the microcontroller without blocking them from the main S12X CPU. The S12X CPU always has priority when the two CPUs access the same resource at the same time. To assure data consistency, the access priority can be controlled by using the hardware semaphores available on the microcontroller.

Interrupts can be routed to the XGATE module in order to decrease the interrupt load of the main S12X CPU. Additionally, up to 8 software triggered channels can be used by the S12X CPU to request software execution on XGATE.

In order to obtain the maximum XGATE CPU speed, the code can be executed from RAM. Because RAM is a volatile memory, XGATE code is being stored into the FLASH memory and copied into RAM after each reset. Having a better RAM access speed and a speed-optimized instruction set, a typical function can run up to 4.5 times faster on XGATE than on the S12X CPU [13]. Because it was designed for quick execution of small code requested by interrupts, the flash memory available for storing XGATE code is limited. For controllers in the S12XD family this limit is 30 kbytes.

3.2 Implementation details

To decrease the communication overhead that could be introduced by computing cryptographic primitives we assigned this task to the XGATE module. In order to evaluate the computational performance of the microcontroller we measured the execution speed of three hash functions: MD5, SHA1 and SHA-256. Measurements were done on S12X and XGATE for different input lengths using both the maximum specified frequency and the overclocked frequency. The measurements presented in tables 1 and 2 show that on average the hash functions were performed approximately 2.12 times faster on XGATE than on S12X. The overclocking also increases the speed with a factor of 2.

Table 1: Performance of S12X in computing MD5, SHA-1 and SHA-256.

Length	Execution time MD5		Execution time SHA1		Execution time SHA-256	
(bytes)	@ 40MHz	@ 80MHz	@ 40MHz	@ 80MHz	@ 40MHz	@ 80MHz
0	$732\mu s$	$371\mu s$	$2.285ms$	$1.144ms$	$5.51ms$	$2.755ms$
1	$736\mu s$	$373\mu s$	$2.290ms$	$1.146ms$	$5.52ms$	$2.760ms$
3	$737\mu s$	$373\mu s$	$2.290ms$	$1.146ms$	$5.52ms$	$2.760ms$
14	$738\mu s$	$374\mu s$	$2.290ms$	$1.148ms$	$5.50ms$	$2.755ms$
26	$739\mu s$	$374.5\mu s$	$2.295ms$	$1.148ms$	$5.49ms$	$2.750ms$
62	$1414\mu s$	$717\mu s$	$4.510ms$	$2.255ms$	$10.86ms$	$5.44ms$
90	$1374\mu s$	$697\mu s$	$4.470ms$	$2.235ms$	$10.80ms$	$5.40ms$

We chose MD5 for building the one-way chains which are needed by the protocol. Due to the small disclosure delay we consider that using MD5 is safe for our scenario. Each chain element will thus be a 128 bit value which is used to perform an HMAC over the message that has be sent at a certain point in time.

One other possible method of building one-way chains is to use block ciphers as hash functions. This can be done by always encrypting a fixed value (e.g. 0) using the previously generated value as the encryption key: $k_i \leftarrow Enc(k_{i-1};0)$. Table 3 shows the performance obtained by S12X in computing some known block ciphers.

All cryptographic computations are done on the XGATE module following a software request. For passing data between the two processing units, a common memory area is used. Each time a hash needs to be computed, the S12X writes the input data in

Table 2: Performance of XGATE in computing MD5, SHA-1 and SHA-256.

Length	Execution time MD5		Execution time SHA1		Execution time SHA-256	
(bytes)	@ 40MHz	@ 80MHz	@ 40MHz	@ 80MHz	@ 40MHz	@ 80MHz
0	312.5μs	156.2μs	1.000ms	500μs	3.155ms	1.578ms
1	314.5μs	157.4μs	1.002ms	501μs	3.155ms	1.580ms
3	314.5μs	157.2μs	1.002ms	502μs	3.155ms	1.580ms
14	316.0μs	158μs	1.004ms	502μs	3.150ms	1.578ms
26	317.5μs	158.9μs	1.004ms	503μs	3.145ms	1.578ms
62	605μs	303μs	1.976ms	988μs	6.24ms	3.125ms
90	592μs	296.5μs	1.962ms	982μs	6.22ms	3.115ms

Table 3: Performance of S12X in computing symmetric primitives @ 80MHz.

Primitive name	Block size	Key size	Rounds	Execution time
Anubis	16 bytes	16 bytes	12	916μs
Blowfish	8 bytes	16 bytes	16	25.2ms
Cast5	8 bytes	16 bytes	16	321μs
Kasumi	8 bytes	16 bytes	8	129.4μs
Skipjack	8 bytes	10 bytes	32	116.2μs
Xtea	8 bytes	16 bytes	32	233.5μs

the common memory area and makes a software request to the XGATE module. While the hash is 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. The XGATE module can signal the end of the function execution by issuing an interrupt to the S12X CPU.

After protocol implementation, the total RAM memory left for storing key chains can hold 1216 elements (16 bytes each). Having this upper limit for M, L and λ have to be determined for best performances based on the bus speed and the desired disclosure delay. If we consider packets of 16 bytes in size the measured bus speed for sending packets is 578 packets/second (one packet each 1.730ms).

For example, if we decide to use three levels to assure authentication over a period of one day with a speed of 578 packets/second, we would have 233 elements on each level and the key disclosure time δ would be 6.8ms. The bus overhead for this situation is 25,2% and the time needed to initialize the key chains is approximately 700ms. Increasing the number of levels to 5 leads to chains of 26 elements so less memory is necessary and the time needed for initialization is reduced to 131ms. The cost of these improvements is a bus overhead of 26.9%. The bus load grows exponentially with the increase in the number of levels used while the disclosure delay depends on the transmission speed and the total communication duration. The duration of the communication also affects the number of elements on each level which is upper bounded to M/L ($1216/L$ in our setting).

4 Conclusions

A protocol for assuring broadcast authenticity on the CAN bus was provided. By this research we hope that we give a first analysis on the feasibility of such a solution in low speed CAN. We studied different trade-offs in order to depict the optimal choice of parameters. In particular we concluded that the main limitation is the bus speed (limited to 128kbps in fault-tolerant CAN), followed by memory and last by computational power. This is also due to the performance of the XGATE co-processor which is about two times faster than the regular S12 processor. In some cases, to reduce memory consumption and to shorten the initialization time, chains with more than three levels were also efficient. The theoretical estimations are entailed by experimental results on the S12X processor, a commonly used automotive grade microcontroller. Current and future work includes extending the results on high end microcontrollers capable of high speed CAN communication.

Acknowledgments. This work was supported by CNCSIS-UEFISCDI project number PNII-IDEI 940/2008 and by the strategic grant POSDRU 107/1.5/S/77265, inside POSDRU Romania 2007-2013 co-financed by the European Social Fund - Investing in People.

References

1. 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.
2. F. Bergadano, D. Cavagnino, and B. Crispo. Individual authentication in multiparty communications. *Computers & Security*, 21(8):719 – 735, 2002.
3. J. Charzinski. Performance of the error detection mechanisms in can. In *Proceedings of the 1st International CAN Conference*, pages 20–29, 1994.
4. Freescale. *MC9S12XDP512 Data Sheet, Rev. 2.21, October 2009.*
5. ISO. *CAN Specification Version 2.0.* Robert BOSCH GmbH, 1991.
6. ISO. *ISO 11898-1. Road vehicles - Controller area network (CAN) - Part 1: Controller area network data link layer and medium access control.* International Organization for Standardization, 2003.
7. ISO. *ISO 11898-4. Road vehicles - Controller area network (CAN) - Part 4: Time triggered communication.* International Organization for Standardization, 2004.
8. 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.
9. L. Lamport. Password authentication with insecure communication. *Commun. ACM*, 24:770–772, November 1981.
10. K. Lemke, C. Paar, and M. Wolf. *Embedded Security in Cars Securing Current and Future Automotive IT Applications.* Springer Verlag, 2006.
11. 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.

12. D. Liu and P. Ning. Multilevel μtesla: Broadcast authentication for distributed sensor networks. *ACM Trans. Embed. Comput. Syst.*, 3:800–836, November 2004.
13. R. Mitchell. *Tutorial: Introducing the XGATE Module to Consumer and Industrial Application Developers, March 2006*. Freescale, 2004.
14. 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.
15. 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.
16. 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.