Secure Broadcast With One-Time Signatures In Controller Area Networks

Bogdan Groza and Stefan Murvay
Politehnica University of Timisoara, Faculty of Automatics and Computers
Email: bogdan.groza@aut.upt.ro, stefan.murvay@gmail.com

***Abstract*—Broadcast authentication in Controller Area Networks (CAN) is subject to real time constraints that are hard to satisfy by expensive public key primitives. For this purpose we study here the use of one-time signatures which can be built on the most computationally efficient one-way functions. We use an enhancement of the classical Merkle signature as well as the more recently proposed HORS signature scheme. Notably these two proposals offer different trade-offs, the first is more efficient in terms of memory while the second is more efficient in terms of verification speed and signature size. Further, both signature schemes can be efficiently paired with time synchronization to reduce the overhead to re-initialize the public keys, which would otherwise require expensive authentication trees. We do outline clear bounds on the performance of such a solution and provide experimental results on development boards equipped with Freescale S12X, a commonly used automotive grade microcontroller. We also benefit from the acceleration offered by the XGATE co-processor available on S12X derivatives which significantly increases computational performances.**

I. Motivation and related work

Controller Area Network (CAN) [7] is a bus wide spread in the field of automotives and commonly present in general purpose automation applications. Traditionally, environments that relied on CAN were isolated in secure perimeters. But things changed nowadays and it is more obvious that control systems (inside a car or not) will be targets of cyberterrorism in the near future. Due to the increased degree of interconnectivity it will be impossible to perfectly isolate any control system from the outside world. A good survey on the subject of security in control systems can be found in [5]. Also, recent research shows how vulnerable cars are to Dolev-Yao adversaries [8]. Therefore, it is clear that controllers will need to authenticate to each other in order to trust the exchanged information. In particular, the importance of assuring security inside a car and on the CAN bus is discussed by Wolf et al. in [22].

It is of our interest here to design and implement at the application layer a broadcast authentication protocol for CAN based on one-time signatures. Symmetric key primitives were successfully used, even in constrained environments such as sensor networks [17], [10], [11]. However using them was feasible only with time synchronization, which is a simple procedure but introduces an authentication delay which needs to be at least several times greater than the synchronization error. Although one can do clever engineering work to improve on this, it will be preferable in some contexts to have immediate authentication. A version of the scheme that achieves immediate authentication is in fact available from Perrig et al. in [16] but this scheme addresses the case in which the Message Authentication Code (MAC) on the message is sent before the key disclosure while the message itself afterwards (allowing to authenticate the message when it is received). This however means that the sender must still wait until the disclosure delay expires in order to authenticate the message. Here by immediate authentication we mean that as soon as a principal knows the value of the message he can broadcast it and its authenticity can be checked by receivers as soon as the authentication tag is received.

To achieve this, there is no other alternative but the use of digital signatures. However, digital signatures are more expensive and require more communication bandwidth. RSA will require thousands of bits for each signature, while elliptic curves can reduce this to several hundreds, still in both cases the computational costs of signing and verifying is very high. To this one will need to add the size of the source code as well as memory requirements which are usually limited in industrial controllers. There is still an alternative to this: the use of one-time signatures which were initially proposed by Merkle in [12], [13]. 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 three on the signer side, which requires even more resources (this resources in fact increase with the number of signed messages). A more recent one-time signature scheme was proposed by Perrig in [15] and a better alternative to it is provided by Reyzin et al. in [18]. A general solution, from which the proposals of Merkle [12], [13] and Lamport [9] can be derived as particular cases, was provided by Bleichenbacher and Maurer in [1] and another work by the same authors studies the optimality of this kind of signatures [2]. Thus, there is good literature available on this subject despite a reduced practical impact.

Contribution. In a previous short paper we made a brief appointment on the use of an improved Merkle one-time signature for this purpose [6]. Here we enhance our previous performance analysis with more details and also implement a more efficient solution based on the HORS signature

scheme [18]. We do find the enhanced Merkle and HORS signatures to offer different tradeoffs, the first is more efficient in terms of memory, while the second is more efficient in terms of signature size and verification time. Indeed, with the HORS signature we exhibit good improvements in the authentication delay. The enhanced Merkle signature also has certain advantages. More concrete, the size of the messages is quite small in most broadcast scenarios since CAN frames carry small data from sensors and actuators (the size of the data field in a CAN frame is actually limited to 64 bits) and this signature allows message recovery, thus small messages can be embedded in the signature. Finally, both signature schemes can be efficiently paired with time synchronization to reduce the overhead to re-initialize the public keys, which would otherwise require expensive authentication trees.

The paper is organized as follows. Section 2 gives an outline of the signature scheme that we use and of the protocol. The theoretical efficiency of this solution is studied in section 2, while in section 3 we proceed on experimental results on S12X microncotrollers which are equipped with XGATE co-processors [14] that can be used to speed up computations. To improve even more we used a hardware random number generator. Section 4 holds the conclusions of our paper.

II. Signature schemes and broadcast protocol

We employ the now classical procedure of using a one-way key chain with time synchronization to commit the public keys that are used to verify the signatures. We stress that the intention is to use these public keys to authenticate the broadcast and not to assure non-repudiation. Because of this we can renounce on the classical structure of Merkle trees to authenticate them, which will be more memory and bandwidth consuming. If one wants to assure non-repudiation at some point, at the cost of extra-memory, then eventually any number of the released public keys could be signed afterwards. The signature schemes are flexible and their parameters can be used to adjust the consumed computational power, memory and bandwidth. These are discussed in detail after the protocols description.

A. The signature schemes

We first review the one-time signature schemes that we are going to use and give an example to underline its efficiency. In the next section we integrate this in the protocol that we are going to use. The generic principle behind both one-time signatures is to apply a simple on-way function, e.g., a hash function, over some input that plays the role of secret key and use the output as public key. However if bits are signed individually this results in an inefficient scheme, not necessarily due to the number of hash computations since these are cheap, but mainly due to the size of the signature itself, e.g., in worst case on hash for each bit. For this purpose, several improvements were proposed in the literature. The enhanced Merkle signature and HORS [18] that we discuss next employ the one-way chains in two highly distinctive fashions, a reason for which we choose to evaluate both of them in out CAN broadcast scenario.

Enhanced Merkle Signature Scheme (EMS). Given one-way function f, signature scheme EMS is a triplet of polynomial time algorithms $Gen, Sign, Ver$ where:

1) Gen is a probabilistic algorithm that takes as input the security level k along with two integers λ, μ and outputs the public-private key pair pk, pv, i.e., $pb = \{(f^{\lambda}(u_{\mu}), f^{\lambda}(v_{\mu}))$,..., $(f^{\lambda}(u_2), f^{\lambda}(v_2)), (f^{\lambda}(u_1), f^{\lambda}(v_1))\}$, $pv = \{(u_{\mu}, v_{\mu}), ..., (u_1, v_1)\} \leftarrow Gen(1^k, \lambda, \mu)$ (here all u_i, v_i are random values of k bits each),
2) $Sign$ is a deterministic algorithm that takes as input the private key pv, a message m of $\lfloor log_2(\lambda) \rfloor \cdot \mu$ bits which can be written as $m = m_{\mu}...m_2 m_1$ (where each m_i has $\lfloor log_2(\lambda) \rfloor$ bits), and outputs a signature s, i.e., $s = \{(f^{\lambda - m_{\mu}}(u_{\mu}), f^{m_{\mu}}(v_{\mu})), ..., (f^{\lambda - m_2}(u_2), f^{m_2}(v_2)), (f^{\lambda - m_1}(u_1), f^{m_1}(v_1))\}$,
3) Ver is a deterministic algorithm that takes as input the signature and the public key and outputs message $m = m_{\mu}...m_2 m_1$ if and only if $\forall i = 1..\mu, f^{\lambda - m_i}(s_i) = f^{\lambda}(u_i), f^{m_i}(s_i) = f^{\lambda}(v_i)$ or $\perp$ otherwise.

Security. The previous scheme is an improvement of the classical Merkle, but as we couldn't find proof for its security we consider to give a proof sketch here. First note that signing each component of the message is independent from another, thus it is sufficient to prove that the adversary is unable to forge any part of the signature. Let Adv be an adversary that manages to forge a signature on some message block m' with non-negligible probability ϵ_{Adv}. We use Adv to make an algorithm that inverts f with non-negligible probability on some target $y = f(x)$. The inverter chooses random $l \leq \lambda$ and random r then flips a bit b. If $b = 0$ then the inverter computes the pair $f^l(y), f^{\lambda}(r)$ which is set as the public key otherwise he computes and sets the public key as $f^{\lambda}(r), f^l(y)$. Now the adversary is allowed to make a query to the signing oracle. Let $\mathrm{Pr}_{\mathrm{Bad}}$ denote the probability that Adv asks for $m_{\mathrm{Adv}} > l$ and $b = 0$, or else $m_{\mathrm{Adv}} < l$ and $b = 1$ which will make the oracle fail to answer and abort. Obviously $\mathrm{Pr}_{\mathrm{Bad}} = 1 - l/\lambda$. Otherwise, the oracle succeeds and the adversary must output the forgery with probability ϵ_{Adv} for some $m' \neq m_{Adv}$. If $m' > m_{Adv}$ as l is also random with probability $(\lambda - l)/\lambda$ we have $m' > l$ and with probability 1/2 we have $b = 0$ which means that we can use the adversary output to invert f with probability $1/2 \cdot (\lambda - l)/\lambda \cdot \epsilon_{\mathrm{Adv}}$. Otherwise, if $m' < m_{Adv}$ with the same probability we can invert f in the second case if $b = 1$. Summing up, the probability to invert the function is non-negligible.

HORS Signature Scheme. Given one-way function f,

signature scheme HORS is a triplet of polynomial time algorithms $Gen, Sign, Ver$ where:

1) Gen is a probabilistic algorithm that takes as input the security parameters l, k and integers λ, μ then generates λ random k-bit values $s_1, s_2, ..., s_\lambda$ then computes $v_i = f(s_i)$ and outputs the public-private key pair pk, pv, i.e., $pb = \{\mu, f(s_1)$,..., $f(s_\lambda)\}$, $pv = (k, s_1, ..., v_\lambda) \leftarrow Gen(1^k, l, \lambda)$,
2) $Sign$ is a deterministic algorithm that takes as input the private key pv and message m then computes $h = hash(m)$ and splits h into k substrings $h_1, ..., h_\mu$ each of $\log_2(t)$ bits and outputs $s = (s_{h_1}, ..., s_{h_\mu})$ (where each h_i is interpreted as an integer index),
3) Ver is a deterministic algorithm that takes as input the signature s, the public key pb and message m the outputs 1 if and only if $f(s'_i) = v_i$ for each i extracted as integer index from $h(m)$.

Security. The security proofs for this scheme can be found in the original paper [18]. We mention here that the security level of this signature scheme is $\mu(\log \lambda - \log \mu - \log r)$ if the adversary obtains r signed messages of its choice.

B. The broadcast protocol

For each of the signature schemes we use a broadcast protocol that relies on one-way key chains. In the case of the EMS signature, the key chain is used to commit future public keys, while in the case of the HORS signature each element of the key chain forms a public key for the signature (this happens in a similar fashion to the BiBa protocol from Perrig [15]).

Time synchronization is loose and is done with synchronization error $\epsilon_{\mathcal{R},\mathcal{S}}$ which is the round-trip time of a handshake between the receiver and the sender. Usually this handshake has two steps as $1.\mathcal{R} \rightarrow \mathcal{S} : N_\mathcal{R}$, $2.\mathcal{S} \rightarrow \mathcal{R} : Sig_\mathcal{S}(t^\mathcal{S}_{sync}, N_\mathcal{R})$ where t_{sync} denotes time on the sender side and $N_\mathcal{R}$ is a nonce chosen by the receiver. However, as we need to keep the synchronization error as small as possible, we will not use a digital signature and instead we will use a MAC which is several order of magnitudes cheaper. By using it, the synchronization error gets to the order of several milliseconds, which is accurate enough for high speeds of the broadcast. Afterwards, the receiver $\mathcal{R}$ can estimate at any point $t_{current}$ that the time on the sender's side $\mathcal{S}$ is $\mathcal{T}_{\mathcal{S},\mathcal{R}}(t_{current}) \in [t^\mathcal{S}_{sync} + t_{current} - t^\mathcal{R}_{sync}, t^\mathcal{S}_{sync} + t_{current} - t^\mathcal{R}_{sync} + \epsilon_{\mathcal{R},\mathcal{S}}]$.

Broadcast with EMS. Given signature scheme EMS and the roles sender $\mathcal{S}$ and receiver $\mathcal{R}$ we define protocol $\text{Broadcast-EMS}_{\mathcal{S},\mathcal{R}}[\lambda, \mu, \delta]$ as the following actions performed by $\mathcal{S}$:

1) *Initialization*: $\mathcal{S}$ generates a key chain by using a random $\mathcal{K}_0$ and computing $\mathcal{K}_n = f(\mathcal{K}_{n-1}), \forall i = 1..n$, then he commits the tip of its top level-chain, i.e., $\mathcal{K}_n$, the disclosure delay δ and the public key obtained by running $Gen(1^k, \lambda, \mu)$,
2) *Commitment*: $\mathcal{S}$ sends at any point in time interval $[t_{start}+i\cdot\delta, t_{start}+(i+1)\cdot\delta-\xi]$ (here ξ is a tolerance value to prevent the sender to commit a MAC to close to the disclosure point which will increase the chance for a receiver to drop the packet) a fresh public key pb generated by using $Gen(1^k, \lambda, \mu)$ and a MAC computed with $\mathcal{K}_{i+1}$ on it, i.e, $\text{MAC}_{\mathcal{K}_{i+1}}(pb)$,
3) *Key Disclosure*: $\mathcal{S}$ sends at time $t_{start} + i \cdot \delta$ the corresponding key from the key chain, i.e., $\mathcal{K}_i$,
4) *Authentic Broadcast*: $\mathcal{S}$ sends at any time in $[t_{start}+i\cdot\delta, t_{start}+(i+1)\cdot\delta-\xi]$ message m as a signature with message recovery $s = Sign(m, pv_{last})$ (here pv_{last} is the most recently generated private key);

and $\mathcal{R}$ respectively:

1) *Initialization*: $\mathcal{R}$ receives the initialization information of the sender, i.e., $\mathcal{K}_n$, the disclosure interval δ and the public key pk,
2) *Time Synchronization*: $\mathcal{R}$ performs a loose time synchronization with $\mathcal{S}$, such that the synchronization error $\epsilon_{\mathcal{R},\mathcal{S}} << \delta$,
3) *Receive Keys and Commitments*: $\mathcal{R}$ receives $\mathcal{K}_i$ and checks if $f(\mathcal{K}_i) = \mathcal{K}_{i-1}$ and discards it otherwise. Any MAC computed with $\mathcal{K}_i$ that is received after $\mathcal{T}_{\mathcal{S},\mathcal{R}}(i \cdot \delta)$ is discarded. Any public key for which there exists a valid MAC and key $\mathcal{K}$ that can verify it is deemed authentic,
4) *Message Verification*: $\mathcal{R}$ runs $Ver(pk_i, sig_i)$ for any valid public key and deems authentic any output different from $\perp$.

Security. The signature scheme was proved to be secure while the security of such protocols based on time synchronization is well known. The informal argument is that from $\text{MAC}_\mathcal{K}(M)$ and $f(\mathcal{K})$ an adversary cannot produce $\text{MAC}_\mathcal{K}(M')$ for any $M' \neq M$ since $\mathcal{K}$ is not known as well as it cannot be found from $f(\mathcal{K})$. 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. For completeness we can give a more formal proof sketch here. 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 the oracle $\mathcal{O}^f$ that computes function f can be replace by a random oracle $\mathcal{O}^R$, which outputs k bits, the proof is straight forward. Assume that the adversary has witness polynomially many queries $p(k)$ to oracle $\mathcal{O}^R$. Suppose at some point the adversary is forced to produce $\text{MAC}_\mathcal{K}(M_{\text{Adv}})$ for some message of its choice. But the adversary knows just $\mathcal{O}^R(\mathcal{K})$ 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 $\mathcal{O}^R$. This means he can guess it and produce a valid MAC only with probability $1/(2^k - p(k))$ - which is negligible.

Broadcast with HORS. Given signature scheme HORS and the roles sender $\mathcal{S}$ and receiver $\mathcal{R}$ we define protocol Broadcast-HORS$_{\mathcal{S},\mathcal{R}}[\lambda, \mu, \delta]$ as the set of following actions performed by $\mathcal{S}$:

1) *Initialization*: $\mathcal{S}$ generates a key chain starting from random $\mathcal{K}_0, ..., \mathcal{K}_\lambda$ and computing $\mathcal{K}_j^i = f(\mathcal{K}_i^{j-1}), \forall i = 1..\lambda, j = 1..\mu$, then he commits the tip each chain, i.e., $\mathcal{K}_i^\mu$,
2) *Authentic Broadcast*: $\mathcal{S}$ sends at any time in $[t_{start} + i \cdot \delta, t_{start} + (i+1) \cdot \delta - \xi]$ message m along with its signature computed with HORS having as secret key input the keys from the current disclosure interval $\mathcal{K}_0^i, \mathcal{K}_1^i, ..., \mathcal{K}_\lambda^i$ (the number of messages signed in each time interval depends on the security level and signature parameters);
3) *Key Disclosure*: $\mathcal{S}$ sends at time $t_{start} + i \cdot \delta$ all the keys from the current interval that were not disclosed as HORS signature (to save some bandwidth, sending these keys can be skipped since the receivers can validate future signatures with previously received keys, but note that this will increase verification time on receivers)

and $\mathcal{R}$ respectively:

1) *Initialization*: $\mathcal{R}$ receives the initialization information of the sender, i.e., $\mathcal{K}_i^\mu, \forall i = 1..\lambda$ and the disclosure interval δ,
2) *Time Synchronization*: $\mathcal{R}$ performs a loose time synchronization with $\mathcal{S}$, such that the synchronization error $\epsilon_{\mathcal{R},\mathcal{S}} << \delta$,
3) *Receive Keys and Commitments*: $\mathcal{R}$ receives keys $\mathcal{K}_i^j$ and checks if $f(\mathcal{K}_i^{j-1}) = \mathcal{K}_i^j$ and discards it otherwise.
4) *Message Verification*: $\mathcal{R}$ runs $Ver(pk_i, sig_i)$ for any signature that is received in the correct time interval and deems authentic any input that is correctly verified.

Security. The security can proved by simply constructing a forger for the HORS signature. In this case a challenger can simply use the public key of the signature to be forged to build key chains and further simulate the broadcast protocol with the hope that an adversary will forge the target signature.

C. Efficiency

We start by analyzing various trade-offs that can be achieved with the enhanced Merkle signature then we compare it to RSA signatures and finally to HORS. The main conclusion is that in general HORS would be more efficient in terms of verification speed and bus load (while it is less efficient in terms of memory requirements) and in the experimental section we provide the computational and communication bounds that we reached for HORS.

Enhanced Merkle Signature. To judge efficiency it is relevant to consider the number of bits that can be signed with a committed public key. Since the length of the chains, i.e., λ, and their number, i.e., μ, is bounded by the computational power in time δ we could write $\lambda = \sigma \cdot \delta / \mu$, where σ denotes computational speed, i.e., the number of function computations per second. Having a fixed σ it is relevant to decide which will be more efficient from a computational point of view: to have larger μ and shorter λ or vice-versa.

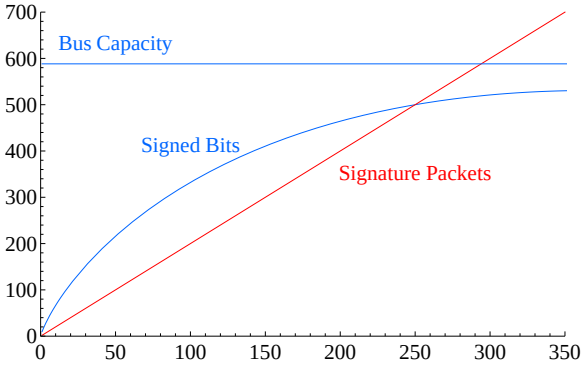

Figure 1. Variation of signed bits and signature size with $\mu \in [0, 350]$

Figure 1 depicts the variation of signed bits and signature size with parameter μ of the signature scheme. The plot is depicted for a speed fixed to 2000 one-way function computations per second, which is around the average of our experimental results presented in the next section. As can be seen, larger μ means more bits can be signed, but require much more bandwidth.

Figure 2 shows the variation of the number of signed bits with computational speed σ and μ. Clearly computational speed cannot compensate enough the decrease in the chain length as it results in division with a logarithmic factor. However, decreasing the chain length results in the same expense of bandwidth as can be seen from figure 1. Thus, higher computational speeds certainly help up to some point when one needs to decrease the length λ in order to allow an increase in μ.

Figure 3 shows the variation of the number of signed bits with computational speed σ and busload B. The plot is given for $\mu \in [0, 500]$ and $B \in [0, 1]$. The maximum number of signed bits is achieved by taking μ equal to half the maximum number of packets that can be send on the bus and then computing λ according to the computational speed and the disclosure delay which is fixed to 1s in this plot. As can be seen, the main limitation for the number of signed bits is given by the bus speed. For example in a fault tolerant CAN with 128kbps, the number of signed bits will not exceed 1.2kbps even if a hash computation does not exceed $100\mu s$. If the bus speed is increased to 1 Mbps, as in high speed CAN, then the number of signed bits can get up to 2.5 kbps. These results hold for the EMS signature, for HORS we discuss the performance in the experimental results section.

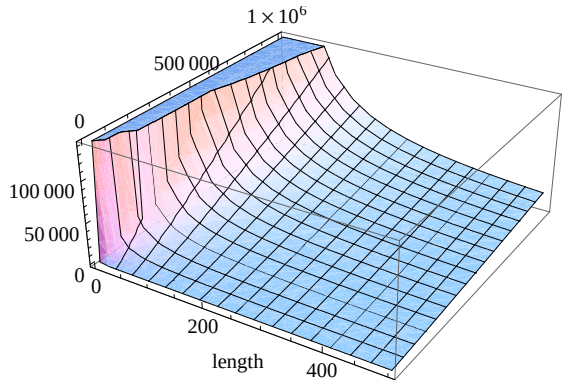

Figure 2. Variation of signed bits with $\sigma \in [0, 1 \times 10^6]$ and $\mu \in [0, 500]$

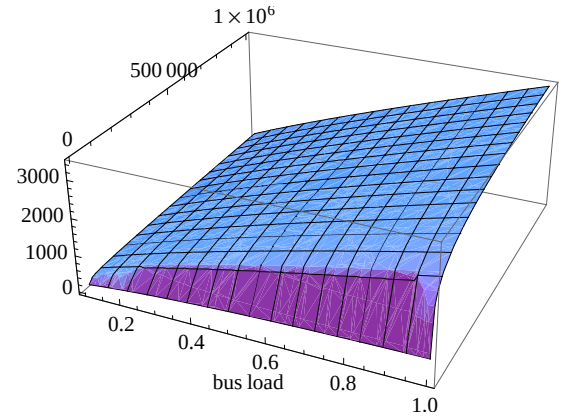

Figure 3. Variation of signed bits with busload (from 0 to 100%) and $\sigma \in [0, 1 \times 10^5]$

Comparison with RSA. A relevant thing about this signature in the way it was presented before is that it allows message recovery. In an environment with constrained bandwidth this becomes relevant with respect to performance. We consider to outline the efficiency of this scheme by a short comparative example with an RSA based signature. If a k bit message is to be signed with fixed λ then $2 \cdot \lambda \cdot \lceil k/log_2\lambda \rceil$ one-way function computations are needed and the signature size is $2 \cdot \lceil k/log_2\lambda \rceil$. Now consider an 1024 bit RSA compared to an MD5. Indeed these primitives are not very strong for today requirements but the proportion gets even worse for the RSA as bigger public keys are used. By taking timings from OpenSSL, on a notebook with an Intel Core2Duo processor at 1.4 Ghz, we get that MD5 is 3670 times faster than the RSA private key operation which is done at signing. Now to sign a 128 bit message assuming $\lambda = 64$ we can process 8 bits at a time which results in $2 \cdot 64 \cdot 16 = 2048$ computations of MD5. That is still almost half the time required by RSA. But indeed it results in a signature that is $2 \cdot 16 \cdot 128 = 4096$ bit long, which is 4 times larger than for RSA. But remember that in our scenario short messages are more common and consider a message is 32 bits. With length $\lambda = 64$ again 8 bits can be processed at a time and we get a signature of the same size as the RSA, i.e., $2\dot{4} \cdot 128 = 1024$, but only $2 \cdot 4 \cdot 64 = 512$ MD5 computations are needed which is 7 times faster than the RSA. For shorter messages, this improves even more, a more detailed analysis is done after the complete description of the protocol. It is commonly known that one can improve on this even more by signing the bits of m only by using the $f^i(u)$ values from the above signature scheme and using the $f^j(v)$ values to sign the sum $\sum_{i=1,\mu}(2^\lambda - m_i)$. In the worst case this will require the same computational costs and size for the signature while in the best case it requires only half the size. To keep the following description simple we leave this just as potential improvement in a practical application.

Comparison with HORS. If we assume the message to be signed is k bits then having length λ for the chains in the enhanced Merkle signatures and HORS then the following constraints hold. The size of the signature, which gives the bus load, is $2k/\log_2 \lambda$ in case of EMS and twice as short in case of HORS, i.e., $k/\log_2 \lambda$. However, in terms of memory HORS requires λ key chains to be stored, while EMS requires only $k/\log_2 \lambda$ key chains which is obviously less. In terms of verification speed HORS is again superior to EMS requiring only $k/\log_2 \lambda$ as opposed to $\lambda k/\log_2 \lambda$ required by EMS. In the experimental section we give practical data on the efficiency of both these schemes.

D. Further improvements: recycling unused keys

Recycling Public keys that were authenticated but unused can be safely used in forthcoming time intervals. The only restriction that must be taken into account is to avoid memory exhaustion attack on the receiver. This is because and adversary may intentionally break the communication channel between $\mathcal{R}$ and $\mathcal{S}$ which will determine the sender to further store public keys until its resources will exhaust. To avoid these a maximum life-span of the public keys can be fixed.

It may be also tempting to recycle unused parts of the chains corresponding to the public keys. If the new tips are authenticated this can be done but combining this with the previous procedures results in an unsafe protocol. For example consider that $\mathcal{S}$ decides to use an unused part of a public key and authenticates it using the top level chain. Now an adversary that has intentionally broken the communication between $\mathcal{S}$ and $\mathcal{R}$ can use the newly committed tips to forge a signature. Because of this, reusing parts of the public keys should be avoided.

III. Experimental results

In order to confirm our theoretical results we made some tests on a 16-bit automotive microcontroller. The Freescale microcontroller (MC9S12XDT512) from the S12X family has 512Kbytes of flash memory and 20Kbytes of RAM. One special feature of this family is the presence of an

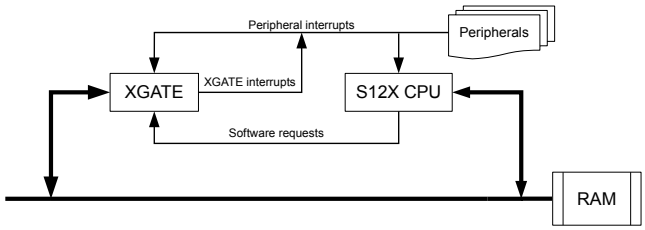

Figure 4. S12X architecture

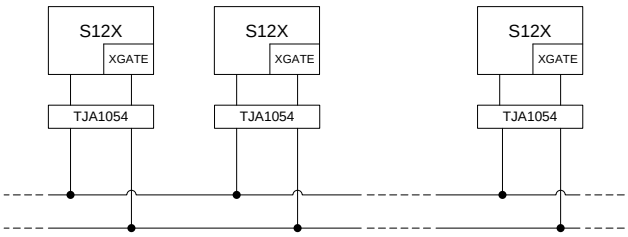

Figure 5. Generic CAN Bus topology

incorporated co-processor called XGATE which was built to reduce the load of the S12X CPU by serving interrupts and software requests. The architecture schematic of the S12X chip can be seen in Figure 4.

For an improvement of the overall performance of the implemented protocols we took advantage of two features available on the chip. On one hand we used a true random number generator (TRNG) implemented with the on-chip analog to digital converter (ADC). On the other hand we used the XGATE co-processor, which can execute operations in parallel to the main S12 CPU at twice the bus speed. This was used to compute cryptographic primitives. Additionally we observed that the operating bus frequency of the microcontroller can be pushed beyond the 40 MHz limit stated by the datasheet without affecting its functionality. We were able to successfully use a maximum frequency of 80 MHz at which the microcontroller was stable.

A. Communication over the CAN bus

CAN bus has a broadcast nature, nodes are connected by a two wire bus topology, as shown in figure 5. Access to the bus is gained with priority based on a message identifier which forms the first part of a frame and has 29 bits in extended frames and 11 bits in standard frames. Other parts of the CAN frame include: a 6 bit control field, 0-64 bits of data, 15 bit CRC and a 2 bit acknowledgment, 1 bit marks the start of frame and 7 bits mark its end. Thus, at most 8 bytes of data can be placed in a CAN frame and they are followed by the 15 bit CRC. 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. In our application setting the S12 development boards are equipped with the Phillips TJA1054 fault tolerant CAN transceiver which allows a sped of 125Kbps.

B. Random number generation

In order to save some computational time, which would have been lost on deriving key material from a master key or by implementing a pseudo-random number generator, we implemented and used a TRNG. As a source of high entropy we rely on a commonly used signal in random number generation: white noise. White noise is defined by Brown [4] as a stationary random process having a constant spectral

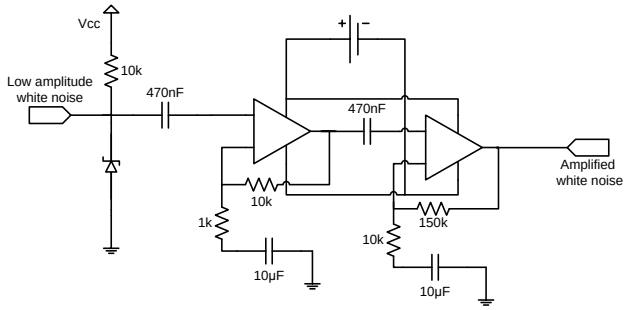

Figure 6. Schematics of the white noise generator that we implemented

density function. This means that the signal contains equal power for each frequency in a fixed bandwidth which will assure that the values are uniformly distributed over the output voltage domain. By sampling the momentary values of the white noise signal we can use the parity of these values to generate random bits.

We have built a white noise generator based on the avalanche noise on a Zener diode caused by the breakdown phenomenon. This appears when the Zener diode is used in reverse polarity mode bringing the PN junction in the reverse breakdown mode. The magnitude of the generated noise was in the order of milli-volts so it had to be amplified. The amplification was done in two steps (with a gain of 11 in the first step and 16 in the second) to bring the voltage level of the signal in the order of several volts. The schematics of the white noise generator used can be seen in Figure 6.

The white noise signal obtained in this way was used as input for one of the ADC channels available on the MC9S12XDT512 microcontroller. This channel was configured as an 8 bit ADC channel making continuous conversions. The parity bit of the conversion result could be used as the random bit but in our case the random bits generated from white noise were biased. The tendency was to generate a zero more often than a one. To solve the bias problem we implemented the simple de-skewing technique proposed by Von Neumann in [20]. Thus we considered pairs of consecutive bits and depending on their values a different action was taken. If the two bits in the pair are identical they

RNG	Test result			
	Passed	Possibly weak	Poor	Failed
White noise	23	2	1	8
RNG CSP	22	2	5	5
.Net Random	24	4	0	6
/dev/urandom/	23	1	1	9
BBS	22	1	5	6

Table II
DIEHARDER STATISTICAL TEST RESULTS

were discarded, a pair of "10" is interpreted as "1" while the pair "01" is interpreted as "0". This method removes the bias problem with a drawback in the generation speed. The speed achieved by this generator was around 2744 bytes/second for a 40 MHz bus frequency and 7835 bytes/second for 80 MHz.

To assess the randomness of the generated random number sequence we used several test batteries: NIST [19] and Dieharder [3]. Multiple sequences were generated in consecutive runs (and also with different controllers on different development boards) and compared to find the number of identical bits. Here we also used for comparison the results obtained for three other RNGs: the Linux cryptographically secure PRNG (/dev/urandom), .Net RNGCryptoServiceProvider and the Blum Blum Shub (BBS) PRNG. The random number sequences tested were 25 Mbytes for each RNG.

We summarize the test results in Table I and II. Table I contains the pass rate of the NIST tests done over the 25 Mbytes sequences which were split in smaller chuncks.

In addition to these tests we also used some of the tests provided by ENT [21] to evaluate if and how the generated sequences converge to expected values. This evaluation gives an image on the distribution of the random numbers generated.

Comparing the results of the five RNGs we observed that our implementation has a similar performance with that of the other random number generators and even sometimes exceeding their performances.

C. Computational performance

Using the main S12X CPU for computing cryptographic primitives, communication and all other necessary operations would lead to a poor performance even if the microcontroller is overclocked at 80 MHz. To compensate for the small frequency we had to reduce the load of the main CPU and we did this by using the XGATE co-processor for executing all cryptographic computations. When a hash for example has to be executed, the main S12X CPU has to issue a software request to the XGATE co-processors. Until the computation is done on XGATE, the main CPU will be free to execute other tasks.

Three well known hash functions have been implemented: MD5, SHA1 and SHA-256. The execution speed for each of these functions was tested both on the S12X CPU and XGATE. The results are presented in Table III for the case of using the maximum documented frequency and the overclocked one. The input for each hash function was equal in length to each specific hash output.

Hash function	Execution time			
	@ 40MHz		@ 80MHz	
	S12X	XGATE	S12X	XGATE
MD5	$730\mu s$	$310.5\mu s$	$367.5\mu s$	$156.8\mu s$
SHA1	$2285\mu s$	$1000\mu s$	$1144\mu s$	$501\mu s$
SHA-256	$5480\mu s$	$3140\mu s$	$2740\mu s$	$1572\mu s$

Table III
PERFORMANCE OF S12X AND XGATE IN COMPUTING HASHES.

Input length	Execution time			
	@ 40MHz		@ 80MHz	
	S12X	XGATE	S12X	XGATE
HMAC-MD5				
64	$5.39ms$	$2.310ms$	$2.695ms$	$1.154ms$
128	$6.02ms$	$2.580ms$	$3.010ms$	$1.288ms$
256	$7.27ms$	$3.110ms$	$3.635ms$	$1.558ms$
512	$9.78ms$	$4.185ms$	$4.890ms$	$2.090ms$
1024	$14.78ms$	$6.33ms$	$7.39ms$	$3.165ms$
HMAC-SHA1				
64	$17.78ms$	$7.79ms$	$8.89ms$	$3.895ms$
128	$19.95ms$	$8.74ms$	$9.96ms$	$4.375ms$
256	$24.25ms$	$10.64ms$	$12.14ms$	$5.32ms$
512	$32.95ms$	$14.44ms$	$16.48ms$	$7.22ms$
1024	$50.4ms$	$22.05ms$	$25.15ms$	$11.04ms$
HMAC-SHA256				
64	$43.1ms$	$24.80ms$	$21.55ms$	$12.42ms$
128	$48.4ms$	$27.90ms$	$24.20ms$	$13.94ms$
256	$59.0ms$	$34.00ms$	$29.55ms$	$17.00ms$
512	$80.1ms$	$46.2ms$	$40.05ms$	$23.15ms$
1024	$122.4ms$	$70.7ms$	$61.20ms$	$35.35ms$

Table IV
PERFORMANCE OF S12X AND XGATE IN COMPUTING MACS.

As the overall authentication overhead is also affected by the commitment of the public keys we also evaluate the time needed to perform a MAC on S12X. We used HMAC together with the three hash functions presented above and a 128 byte key. Table IV holds the execution time we obtained for different message sizes.

D. Protocol performance

As shown in the previous section (Table III), the computation of one MD5 is done in $156.8\mu s$ on XGATE at a frequency of 80 MHz. With this speed, considering data blocks of 64 bits with $\lambda = 64$ and $\mu = 47$ (bounded by the computational speed) we get a bus load of around 16% and approximately 286 bits can be authenticated in one second. For a bus load of 50%, having $\lambda = 21$ and $\mu = 147$, the authentication speed increases to 652 bps. To reach an authentication speed of 1000 bps we can use $\lambda = 10$ and $\mu = 294$ but at the cost of a bus load of 100%. This may not seem much, but it allows a flexible tradeoff between the length and the number of signed messages. For example, in the first case, at a bus load of only 16% a number

Test	RNG				
	White noise	RNG CSP	.Net Rand	/dev/urandom/	BBS
Frequency	0.9880	0.9800	0.9840	0.9960	0.9840
Block Frequency	0.9880	0.9920	0.9880	0.9960	0.9960
Cusum Forward	0.9880	0.9840	0.9880	0.9960	0.9800
Cusum Reverse	0.9840	0.9920	0.9840	0.9920	0.9880
Runs	0.9960	1.0000	0.9960	0.9920	0.9840
Longest Run	0.9840	0.9840	0.9960	0.9880	0.9960
Rank	0.9920	0.9960	0.9960	0.9960	0.9880
FFT	0.9920	0.9880	0.9840	0.9920	0.9840
Non Overlp.Tpl.	0.9892	0.9885	0.9886	0.9886	0.9904
Overlapped Template	0.9880	0.9960	0.9840	0.9920	0.9920
Approx. Entropy	0.9960	0.9800	0.9840	0.9840	0.9880
Serial 1	0.9880	0.9720	0.9920	0.9920	0.9920
Serial 2	0.9840	0.9880	0.9920	0.9880	0.9960
Linear complexity	0.9760	0.9800	0.9840	0.9920	1.0000

Table I
THE PASSING RATE OF THE NIST STATISTICAL TESTS

of 47 messages of 8 bits can be signed in each second which is enough to hold critical data from analog-to-digital converters, etc., while 74% of the bus is free and can be used for other non-critical tasks. This amount of messages cannot be signed with a public key primitive such as the RSA, which requires hundreds of milliseconds on S12. This contrast shows the efficiency of the employed mechanism.

For implementing the HORS protocol we looked for a suitable way of adapting it for devices with lower computational powers. We adapted our setup by adding a master node which has the sole purpose of authenticating messages that are broadcasted on the CAN bus. To allow this, each of the other participants to the communication (which will be called slave) has to share a secret key with the master. In this way, when a slave wishes to send an authenticated message, it will put one frame on the bus containing the message and a counter, followed by another frame which will contain a MAC computed using the preshared key over the message-counter pair. The master checks the authenticity of the message using the key associated with the sender ID and if it succeeds it will send the HORS authentication information to all nodes. We chose to use a master node because HORS involves using a high number of key chains which would need a considerable amount of memory for storing. Since the bus nodes are constrained even in what regards the available memory, the most cost effective solution would be to have only one device with higher computational speed and storage that plays the role of a master.

Even in this master-slave setup it is not an easy task to find a suitable parameter combination that will consume as little as possible from the slave constrained resources and in the same time allow for a good communication speed. We tested a setup that uses $t = 1024$ key chains of $k = 48$ bits each as the signature chain and MD5 as our f function. The result of the MD5 is split into $k = 7$ substrings of 10 bit each ($\log_2(t)$). As the master node, we used a laptop (Intel Core2Duo CPU T7700@2.4GHz) along with a CANcardXL (PCMCIA device) and CANcab 1054mag to enable the PC to communicate over CAN. Using these parameters in the master-slave setup described above we were able to reach an authentication delay of 30ms when setting the S12X chip frequency to 40MHz without employing the XGATE coprocessor. By increasing the frequency to 80MHz and using XGATE to perform the cryptographic computations this delay can be decreased to around 15ms. Alternatively, this leads to 30–60 authentic packets each second while the size of each packet is not bounded by any parameters (different to the case of the EMS scheme) except for bus speed.

IV. CONCLUSIONS

One-time signatures prove to be an efficient mechanism for assuring immediate authentication in CAN networks. This is more prominent in the case of messages with reduced size. The main purpose of our work was to assess which are the limitations of such a solution in CAN networks and our experimental results showed up to several kilo-bits of authenticated traffic can be achieved while the delay at which messages are authenticated can be lower to tens of milliseconds. In our experimental results the computational performance was also boosted by the XGATE co-processor which is two to four times faster than the regular S12 chip. We also improved on this as one-time signatures require fresh random values and we take them from hardware RNG, thus avoiding a key derivation process or a software PRNG which will require more computational time. As future work we intend to make a full scale implementation of the protocol described here both on the S12X controller and on a high end automotive controller with better computational speed and high speed CAN.

ACKNOWLEDGMENT

This work was partially supported by national research grant PNCDI PN II 940/2009 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] D. Bleichenbacher and U. Maurer. Directed acyclic graphs, one-way functions and digital signatures. In Y. Desmedt, editor, *Advances in Cryptology CRYPTO 94*, volume 839 of *Lecture Notes in Computer Science*, pages 75–82. Springer Berlin / Heidelberg, 1994.

[2] D. Bleichenbacher and U. Maurer. On the efficiency of one-time digital signatures. In K. Kim and T. Matsumoto, editors, *Advances in Cryptology ASIACRYPT '96*, volume 1163 of *Lecture Notes in Computer Science*, pages 145–158. Springer Berlin / Heidelberg, 1996.

[3] R. Brown, D. Eddelbuettel, and D. Bauer. Dieharder: A random number test suite.

[4] R. G. Brown. *Introduction to random signal analysis and Kalman filtering*. John Wiley & Sons, 1983.

[5] 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.

[6] B. Groza and S. Murvay. Secure broadcast with one-time signatures in controller area networks. In *Availability, Reliability and Security (ARES), 2011 Sixth International Conference on*, pages 371–376. IEEE, 2011.

[7] 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.

[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. Constructing digital signatures from a one-way function. Technical report, Technical Report CSL-98, SRI International, 1979.

[10] 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.

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

[12] R. Merkle. Secrecy, authentication, and public key systems. 1979.

[13] 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.

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

[15] A. Perrig. The BiBa one-time signature and broadcast authentication protocol. In *Proceedings of the Eighth ACM Conference on Computer and Communications Security (CCS-8)*, pages 28–37, Philadelphia PA, USA, 2001.

[16] 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.

[17] 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.

[18] L. Reyzin and N. Reyzin. Better than biba: Short one-time signatures with fast signing and verifying. In *Proceedings of the 7th Australian Conference on Information Security and Privacy*, ACISP '02, pages 144–153, London, UK, 2002. Springer-Verlag.

[19] A. Rukhin, J. Soto, J. Nechvatal, M. Smid, E. Barker, S. Leigh, M. Levenson, M. Vangel, D. Banks, A. Heckert, J. Dray, , and S. Vo. A statistical test suite for random and pseudorandom number generators for cryptographic applications, nist special publication 800-22, Apr. 2010.

[20] J. von Neumann. Various techniques used in connection with random digits. In A. S. H. et al., editor, *The Monte Carlo Method*, volume 12, pages 36–38. National Bureau of Standards, Applied Mathematics Series, 1951.

[21] J. Walker. Ent - a pseudorandom number sequence test program.

[22] M. Wolf, A. Weimerskirch, and C. Paar. Secure in-vehicle communication. In *Embedded Security in Cars Securing Current and Future Automotive IT Applications*. Springer Verlag, 2006.