CANAuth - A Simple, Backward Compatible Broadcast Authentication Protocol for CAN bus

Anthony Van Herrewege, Dave Singelee, Ingrid Verbauwhede
firstname.lastname@esat.kuleuven.be

***Abstract*— The Controller-Area Network (CAN) bus protocol [1] is a bus protocol invented in 1986 by Robert Bosch GmbH, originally intended for automotive use. By now, the bus can be found in devices ranging from cars and trucks, over lightning setups to industrial looms. Due to its nature, it is a system very much focused on safety, i.e., reliability. Unfortunately, there is no build-in way to enforce security, such as encryption or authentication.**

In this paper, we investigate the problems associated with implementing a backward compatible message authentication protocol on the CAN bus. We show which constraints such a protocol has to meet and why this eliminates, to the best of our knowledge, all the authentication protocols published so far.

Furthermore, we present a message authentication protocol, CANAuth, that meets all of the requirements set forth and does not violate any constraint of the CAN bus.

***Keywords*—CAN bus, embedded networks, broadcast authentication, symmetric cryptography**

I. Introduction

The Controller-Area Network (CAN) bus protocol [1] is a bus protocol invented in 1986 by Robert Bosch GmbH, originally intended for automotive use.

CAN bus nodes are all connected to the same, shared bus line. The CAN bus is wired such that a 0-signal is dominant over a recessive 1-signal. These dominant and recessive signals are used for a CSMA/CA scheme. An arbitration scheme using priority resolution is used to decide which node is allowed to transmit data over the bus. The lower a node its ID (and thus the more dominant 0s it sends during bus arbitration), the higher its priority. Using this scheme makes the CAN bus fit for use as a real-time communication bus.

During transmission, a node continuously checks the signal on the bus and compare this with the signal it is transmitting. If a mismatch occurs, an error is raised, except during arbitration, in which case the node will just stop transmitting. After each message, a CRC is transmitted and receiving nodes will raise an error in case a mismatch occurs. To prevent broken nodes from continuously disturbing and invalidating messages, internal error counters are kept which, depending on their value, disable the right of a node to raise errors. This guarantees that communication over the CAN bus is very reliable, since, even with broken nodes raising errors, messages will eventually be transmitted successfully.

One year after its introduction, Philips presented the first CAN bus controller and, in time, the protocol was used more and more in non-automotive machines. By now, the bus protocol can be found in a wide range of devices: from cars and trucks, over lightning setups to industrial looms, printers, freezers and trains.

Due to its nature, the design of CAN bus is very much focused on safety, i.e., reliability. Unfortunately, there is no build-in way to enforce security, such as encryption or authentication. This leads to many possible attacks, as demonstrated by Koscher et al. [2] and Checkoway et al. [3]. Examples of such attacks are controlling brakes, remotely starting the car and controlling the air conditioning.

In this paper, we investigate the possibility of implementing a backward compatible message authentication protocol on the CAN bus. We show which constraints such a protocol has to meet and why this eliminates, to the best of our knowledge, all the authentication protocols published so far.

In Section II, we very briefly show some of the work that is already been done on broadcast authentication protocols. Section III gives a detailed overview of the requirements such a protocol should meet and the constraints it should honor for it to be usable on the CAN bus. Then, in Section IV, we present our proposal for such an authentication protocol. In Section V, an adversarial model is defined and the security properties of the protocol are investigated. Finally, we present our conclusions in Section VI.

II. Previous Work

In the past, different protocols have been published on how to archieve authentication in broadcast networks, some of those even aimed at lightweight embedded networks. The next few paragraphs give a very concise overview of some of those protocols.

The TESLA protocols are a family of related lightweight authentication protocols, relying on delayed key disclosure to guarantee message authenticity. The original TESLA protocol was published by Perrig et al. in 2000 [4], [5]. This protocol is designed to provide authenticated broadcast capabilities. Subsequently, Perrig et al. presented μTESLA [6], a modification of the original TESLA protocol, aimed at sensor networks, with severe contraints placed on computation and storage capabilities.

Other protocols have been published for use in broadcast networks, such as a symmetric solution by Gennaro and

Rohatgi [7] and Rohatgi's improved protocol [8].

Unfortunately, all of these protocols have certain characteristics which violate the constraints set in Section III, such as requiring challenge-response communication, introducing delays (the main problem with TESLA and μTESLA) or needing to transmit large amounts of data (the main problem of the solutions by Gennaro and Rohatgi), all of which are not acceptable in a CAN bus network.

III. Problem Overview

In this section we explain the requirements and constraints for a backward compatible, lightweight message authentication protocol on CAN bus. This should make it clear why none of the published protocols so far are usable on the CAN bus. Furthermore, we show how we will we meet those requirements and constraints with our proposed CANAuth protocol.

A. Authentication Protocol Requirements

The basic requirements a message authentication protocol should provide are:

Message authentication
: The authenticity of messages should be provable. The exact origin of the message is not important as long as it is guaranteed that it was sent by a trusted node.

Replay attack resistance
: Replaying previously sent authenticated messages should lead to those messages being discarded by the verifying nodes.

Group keys
: It should be possible for a group of related messages to be authenticated with the same key[1]. This reduces the size of key storage needed.

Backward compatibility
: A node employing the authentication protocol has to be able to authenticate its messages without disturbing the workings of any incompatible nodes. It should be possible to connect a number of nodes supporting the authentication protocol to an existing bus without having to do any reconfiguration of the existing nodes.

B. Restrictions Due To CANBus

Although many authentication protocols exist, the CAN bus presents a few peculiar problems. Due to the fact that we want a backward compatible authentication protocol, following restrictions have to be taken into account:

Hard real-time
: CAN bus systems are often employed in environments where hard real-time constraints apply, e.g., cars. Therefore, message transmission and processing times should not be significantly influenced by the authentication protocol. All authentication data needs to be send along with its message, so as not to disturb the real-time response capabilities of the system.

Message length
: A single message on the CAN bus can contain between 1 and 8 bytes of data. Any extra authenticated data needed by the authentication protocol somehow has to be transmitted along with the message it belongs to.

Message IDs
: Each type of message on the CAN bus is associated with a certain ID, e.g. ID 1 = temperature of location L_1, ID 2 = humidity of location L_1, ... An ID consists of 11 bits[2] and due to the fact that an ID is coupled to a very specific message content, most of the IDs will already be taken in an existing network. This means one can not go around adding extra IDs with some new content type.

Unidirectional communication
: A CAN interface sending messages has no ID for itself and all communication is broadcast. Due to this there is no notion of bi-directional communication between nodes, since CAN interfaces have no ID. The only bi-directional "communication" possible between multiple nodes is though an error flag. Even then, a transmitting node can not find out which receiving node raised the error.

The first requirement, *Message authentication*, can be met by attaching a message authentication code (MAC) to a message. Due to the *Hard real-time* restriction, the employed MAC algorithm needs to be fast. Ideally, it should be possible to start processing for verification as soon as part of the authentication data is received. One algorithm satisfying these requirements is HMAC [9], [10], assuming the hash function employed therein is fast.

The *Replay attack resistance* requirement can then be met by incorporating some counter value in the MAC calculation. The same counter value should never be reused however, which could lead to problems if a counter of limited length is used and a large amount of authenticated messages are sent. Furthermore, the less state that has to be saved across subsequent system restarts, the better. A solution for this is the establishment of a session-key during system startup. A limited length counter can then be made to work in our benefit, since it allows the system to gauge when a new session-key should be established.

Due to the restriction on *Message IDs* and *Unidirectional communication*, nodes can not send a message back in response to some message they receive. That would require the creation of a large amount of new IDs, each with a specific new content type such as ID 5 = response of node $\mathcal{N}_i$ to messages with ID 1. Thus, if n nodes want to be

[1]Note that this is slightly different from the usual use of the term *Group key*, which normally implies a key shared between a group of users. In this case the key is shared between a group of messages.

[2]When using extended CAN, IDs are 29 bits long.

able to respond to m different message IDs, $n \cdot m$ IDs have to be created. Given the large number of different message content types, that is not possible as it would quickly lead to exhaustion of all the available 11 bit IDs. Furthermore, it would require reconfiguration of existing nodes on a bus each time a new node is added. In case bi-directional communication should prove necessary, a (backward compatible) workaround will need to be devised.

Supporting the *Group keys* requirement is straightforward using options made available by the CAN protocol. In general, one can program a CAN interface with a number of *acceptance codes*. These codes tell the interface to listen to messages with an ID matching any of the *acceptance codes*. It is also possible to specify *acceptance masks*, allowing a single *acceptance code* to match multiple IDs. By linking keys to these $\{$*acceptance code*, *acceptance mask*$\}$ sets, a *group key* setup can be achieved. We define a group of related messages $\mathcal{G}_i$ as the collection of all messages with IDs matching the pair $\{$*acceptance code*, *acceptance mask*$\}_i$.

The last requirement, *Backward compatibility* presents the biggest problem. For one, attaching authentication data by concatenating it with an existing CAN message is impossible, since that will violate the *Message length* constraint. Neither can the authentication data be put inside a CAN message, since this will decrease the already very limited maximum message size. A third option would be to sent one long data packet over multiple messages. However, this would reduce the real-time capabilities of the system.

C. Transmission with CAN+

One solution to all this is sending the authentication data through an out-of-band channel. This can be achieved with the CAN+ protocol [11]. Using this protocol, extra bits can be inserted in between the sampling points of a CAN bus interface. A graphical presentation of how this works, is shown in Fig. 1.

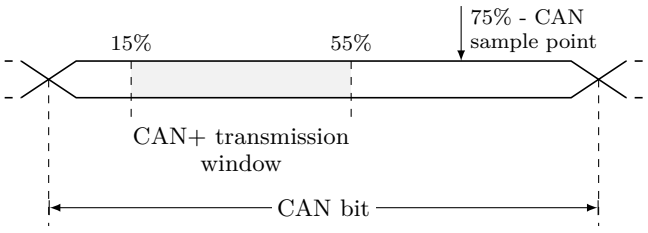

Fig. 1. Timeframe during which the CAN+ protocol can insert extra data into a CAN bit, without disrupting the working of regular CAN bus controllers.

The number of data bits that can be transmitted this way is limited by the maximum attainable clock speed of the CAN+ interface. Ziermann et al. report that on a 1 MHz CAN network, they can transmit 15 CAN+ bytes for each CAN byte, using an FPGA running at 300 MHz. At lower CAN bus speeds, this number increases linearly as:

$$\frac{\text{CAN+ data bits}}{\text{CAN data bit}} = \frac{1\text{ MHz}}{f_{\text{bus}}} \cdot 16 - 1.$$

The loss of one CAN+ data bit is due to the need to send a start bit at the beginning of every CAN+ transmission. On a 100 kHz CAN network, it is thus possible to transmit up to 159 CAN+ bits for each CAN bit.

Since it should be possible to authenticate a message irrespective of its length or the bus speed of the CAN bus network, authentication data should be limited to 15 bytes. This allows one to send all necessary authentication data in the worst case scenario, i.e. as part of a 1 byte CAN message on a 1 MHz CAN bus. Due to this restriction on authentication data length, the use of public-key (and identity-based) cryptography is not possible, due to its large key size requirement compared to symmetric cryptography.

The restrictions and requirements an authentication protocol has to adhere to on the CAN bus make any published protocols, to the best of our knowledge, unusable.

D. Problems with multi-node transmissions

One could devise a system whereby multiple nodes transmit during the CAN+ timeframe, which would then allow bi-directional communication. This would violate the hard real-time constraint as soon as a certain number of nodes needed to sent data, since one would still need multiple messages to do that. During a key establishment, such a breach of constraints could be disregarded though. However, in the next paragraph, we prove that any protocol that requires bi-directional communication between nodes, will impose a speed limit on the CAN bus.

The CAN bus protocol uses a Carrier Sense Multiple Access with Collision Detection (CSMA/CA) protocol during transmission, whereby bit collisions are to be detected within a bit its transmission window. Assume one wanted to sent a minimum of one data bit with the CAN+ protocol per CAN bit. Each CAN+ transmission begins with a start bit, so there would have to be two bit transmissions during the CAN+ timeframe. This timeframe occupies maximum 40% of a CAN bus bit transmission timeframe. Thus, in this case, the CAN+ protocol would need to work at a frequency at least $2 \cdot \frac{1}{0.40} = 5$ times faster than the CAN protocol.

The CAN bus standard guarantees that for a bus length of a maximum of 30 meter, CSMA/CA, and thus the CAN bus protocol, can work at a maximum bus speed of 1 MHz. However, since we want CSMA/CA to function during the CAN+ transmission, which needs to work at a frequency at least 5 times as fast as the CAN bus, the maximum CAN bus speed for the given bus length is only 200 kHz.

Furthermore, at this rate, of the 16 CAN + bits sent during a CAN byte transmission, only 8 CAN + bits are usable for data, since the other 8 are needed as start bits. The more data bits one wishes to sent during the CAN+ transmission window, the lower the CAN bus speed needs

to be if one wishes to adhere to the maximum speed of 1 MHz for CAN+.

Of course, it is possible to increase the speed of CAN+ in our example to 5 MHz, in which case CAN can run at 1 MHz. However, if so, the maximum bus length would be reduced as well, to allow for timely collision detection during the CAN+ transmissions.

Thus, if one requires bi-directional communication with the help of CAN+, either the maximum bus length or the CAN bus speed needs to decrease.

IV. CANAuth Protocol

In this section we propose a simple authentication protocol, CANAuth, that meets all of the requirements set forth in Section III and does not violate any of the constraints.

A. Data Transmission and Frame Format

As suggested in Section III, authentication data is transmitted out-of-band with the CAN+ [11] protocol, which gives us a maximum length of 15 bytes for the authentication message. The bytes are subdivided in fields as shown in Fig. 2.

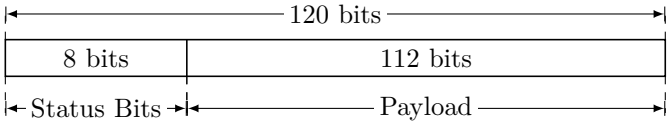

Fig. 2. The CANAuth data frame fields. The data frame consists of the first 15 CAN+ bytes (= 120 CAN+ bits) of a CAN message.

The first eight bits are used as status flags. Currently, only the first two of these bits are used, with the remaining six available to be used in future versions of the protocol. The remaining 112 bits are used to either transmit key establishment or signature data. In the following subsections, more information is given about these two possibilities.

B. Key Establishment

The key establishment protocol described here requires one or more pre-shared 128 bit keys to be available on each CANAuth node. Each group of related messages $\mathcal{G}i$ should get a pre-shared key $\mathcal{K}p_i$ assigned during development of the CAN bus design[3]. We assume that the keys are stored in some tamper-proof storage and are unable to be queried by anyone but the node itself.

The node responsible for transmitting messages $\mathcal{G}i$ is to set up the key for that group. In case multiple nodes transmit messages $\mathcal{G}i$, the key established by the node transmitting with the lowest ID for that group gains precedence.

[3]Even better would be if all 'devices' of the same type (i.e. all Brand X Model Y cars) all contained different keys, programmed during production. This to prevent one $\mathcal{K}p_i$ being found from leading to a security breach of all devices of the same type.

To prevent replay attacks, key establishment works in two phases. First, the appropriate node (as per the rule defined above) transmits a message on the CAN bus with an attached CAN+ message of the form shown in Fig. 3.

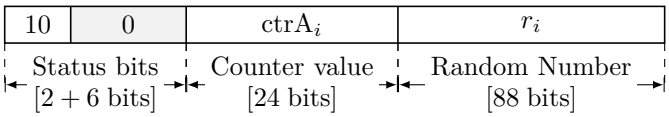

Fig. 3. The CANAuth data frame fields during the first part of key establishment.

The first bit of the frame is set to one to signal that key establishment is taking place. The second bit is set to zero, signalling that this message is the first of two needed for key establishment. The next six bits are unused and should be set to zero. The actual payload of the message is a 24 bit counter and an 88 bit random number.

With the counter value ctrA_i and the random number r_i all nodes possessing the pre-shared key $\mathcal{K}p_i$ generate the session key $\mathcal{K}s_i$ for messages $\mathcal{G}_i$ using HMAC [9]. Implementations of CANAuth are free to use whichever hashing algorithm for HMAC that is deemed strong enough. Depending on the hashing algorithm used, the HMAC output will be longer than 128 bits. The session key $\mathcal{K}s_i$ consists of the 128 LSB of the HMAC output and is generated as follows:

$$\mathcal{K}s_i = \text{HMAC}(\mathcal{K}p_i, \text{ctrA}_i \parallel r_i) \bmod 2^{128}. \qquad (1)$$

The addition of a counter in the message prevents an adversary from isolating the trusted key establishment node from the bus and setting up a session key with the other nodes on the bus, using a previously transmitted random number. Even though the adversary would not know the session key $\mathcal{K}s_i$, he could then transmit recorded authenticated messages. Thus, to prevent this attack, before any node accepts the session key $\mathcal{K}s_i$, it should verify that the random number did in fact originate from a trusted node and that it has not been used before.

There are two checks to guarantee this. First, each node on the bus should store the last validated counter value ctrA_i for each $\mathcal{G}_i$ in non-volatile memory. New key establishment messages are only accepted when the transmitted counter value is higher than the stored value. Second, as the next step in the key establishment, the key establishment node transmits a second message, using the format shown in Fig. 4.

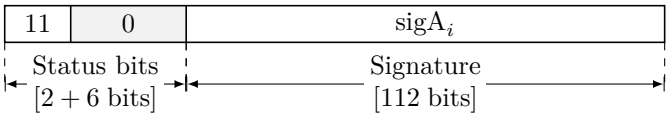

Fig. 4. The CANAuth data frame fields during the second part of key establishment.

In this case, the first and second bits are both set to one, to signal that this is a message to authenticate the key establishment. The next six bits are again unused and should be set to zero. Following that is a 112 bit signature generated as follows:

$$\text{sigA}_i = \text{HMAC}(\mathcal{K}s_i, \text{ctrA}_i \parallel r_i) \bmod 2^{112}. \quad (2)$$

All nodes participating in the establishment protocol can now verify that the sending node knows the session key and is thus trustworthy and that ctrA_i and r_i where not tampered with. These nodes, including the transmitting node, set their stored value ctrA_i to whatever value was transmitted in the first key establishment message.

An adversary could still isolate the key establishment node from the bus and get valid pairs of key establishment messages, but to keep the machine in working condition, will have to forward those messages to the rest of the bus anyway, due to reasons we explain later.

The addition of the counter to the key establishment protocol does not only have benefits though. Once the counter is at its maximum value, listening nodes will not accept new session keys anymore, since the transmitted counter value can not be higher than the stored counter value. This limits the maximum lifetime of any machine utilizing the CANAuth protocol. That is why we propose the use of a 24 bit counter field, which allows $2^{24} = 16\,777\,216$ session key setups, enough to set up a new session key once every minute for ± 32 years.

In case an error is raised in response to any of the two key establishment messages, the key establishment node will restart the protocol from the first message.

C. Message Authentication

Once a session key has been established, messages $\mathcal{G}_i$ can be authenticated. The frame format for authentication data is shown in Fig. 5.

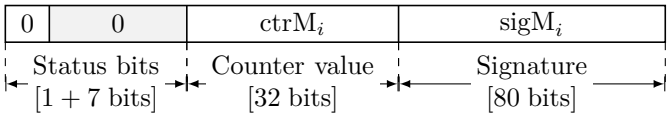

Fig. 5. The CANAuth data frame fields during message authentication.

The counter value ctrM_i is there to prevent replay attacks. Nodes should keep a local copy of the value of ctrM_i encountered in the last valid message. Authenticated message should only be accepted when ctrM_i is greater than the stored value. Due to this rule, the session key $\mathcal{K}s_i$ needs to be renewed each time ctrM_i is about to wrap around back to 0. For every message $\mathcal{G}_i$ a node transmits, ctrM_i should be increased by at least 1.

The second part of the authentication data frame is the signature, which is obtained by taking the 80 LSB of the HMAC of the counter value ctrM_i and the CAN message data msg_i under the session key $\mathcal{K}s_i$, i.e.:

$$\text{sigM}_i = \text{HMAC}(\mathcal{K}s_i, \text{ctrM}_i \parallel \text{msg}_i) \bmod 2^{80}. \quad (3)$$

D. Handling of Invalid Authentications

On a CAN bus system, every node can transmit an error frame at any time, which invalidates the message that is being transmitted. Nodes keep track of the number of errors on the bus. A node that is receiving data will increase its receiving error counter (*REC*) by 1 for each erroneous message, except if it signaled the error itself, in which case its *REC* is increased by 8. For every successfully received message, the *REC* is decreased by 1.

To prevent a faulty node from invalidating every message on the bus, nodes are allowed to signal errors as long as their *REC* is below 127. Once a node its *REC* value is higher, it has to abstain from signalling errors until enough messages have been successfully received to decrease its *REC* below 127.

CANAuth leverages this error handling capability by utilizing the counters in much the same way. Whenever a node can not successfully validate a message, CANAuth will make that node raise an error and increase its REC by 8. Whenever an error is raised, every CANAuth compatible node should discard the corresponding message, even if a node does not detect an error and can successfully verify the attached signature.

E. A Note on Hardware Implementation Speed

The latest a message can be rejected on the CAN bus is by signalling an error frame after the transmission of the ACK delimiter bit. The end of the ACK delimiter bit is 3 CAN bit lengths after the last CRC bit has been send. Thus, to be able to reject messages with invalid CANAuth data, the CANAuth controller should be done verifying the signature by then.

The transmission speed of CAN+, over which CANAuth data is transmitted, should be at least 40 times that of the CAN bus[4]. Since the CAN+ bits need to be sampled, the internal clock of a CAN+ controller will need to be at least twice as fast as that, so for every transmission of a CAN bit, there are 80 clock cycles on the CAN+ controller.

For a CAN message of 1 byte, there are 18 CAN bit lengths between the transmission of the last CAN data bit and the transmission of the ACK delimiter bit. Furthermore, the last CANAuth bit is transmitted after 55% of the duration of a CAN bit transmission [11]. Thus, there are at least $(18+0.45) \cdot 80 = 1\,476$ clock cycles available for the CANAuth implementation to calculate and verify the HMAC signature, assuming a CAN message with 1 byte of data and 16 CAN + bits per CAN bit. That should be

[4]CAN+ can only utilize a maximum of 40% of the duration of a CAN bit for transmission. During this time, for CANAuth to work, at least 16 bits need to be transmitted, thus $f_{\text{CAN+}} = \frac{16}{0.4} \cdot f_{\text{CAN}} = 40 \cdot f_{\text{CAN}}$.

more than sufficient for any hardware implementation of a hash function to calculate its result.

V. Security Properties

The requirements and workings of the CANAuth authentication protocol have been explained. In this section, we define a simple model for an adversary $\mathcal{A}$ upon which we can base ourselves when discussing the security of our authentication protocol. Furthermore, we show how certain properties of the CAN bus make attacks a lot harder.

A. Adversarial Model

The adversary $\mathcal{A}$ has access to all the data that is transmitted over the CAN bus. Furthermore, $\mathcal{A}$ has physical access to the bus and the nodes. Thus, $\mathcal{A}$ is capable of setting up a Man-in-the-Middle (MitM) attack. Although $\mathcal{A}$ has access to the nodes, it reasonable to assume $\mathcal{A}$ does not possess the technology to set up invasive attacks on the nodes, thereby allowing access to e.g. tamper-proof storage.

We furthermore assume that all pre-shared and session keys are stored in tamper-proof to and from which only the node can write and read. Counter values should be stored in tamper-proof memory as well, but are not required to be shielded from reading by an outsider, since their values are public anyway.

B. Denial-of-Service Attacks

One important class of attacks are Denial-of-Service (DoS) attacks. It is trivial to invent methods to set up a DoS attack on the proposed protocol. Since such attacks will always somehow involve disturbing the signal on the CAN bus, it would be just as simple, if not simpler, for $\mathcal{A}$ to connect the bus to ground at strategic times to create an error condition. Since it is impossible to defend against such an attack, we will ignore any kind of attack that can be reduced to a simple DoS attack.

An example of one such attack, goes as follows. We have a CAN bus with two nodes, $\mathcal{N}_a$ and $\mathcal{N}_b$, which both know a pre-shared key $\mathcal{K}p_1$. During the first key establishment message, $\mathcal{A}$ executes a straightforward MitM attack:

$$\mathcal{N}_a \quad \xrightarrow{r_1} \quad \mathcal{A} \quad \xrightarrow{r_1'} \quad \mathcal{N}_b.$$

The result is that nodes $\mathcal{N}_a$ and $\mathcal{N}_b$ generate different session keys, and thus $\mathcal{N}_b$ will find the key establishment authentication message invalid. Thus, $\mathcal{N}_a$ will restart the key establishment. The attacker $\mathcal{A}$ learns nothing however and could have achieved exactly the same result by raising an error frame when the key establishment authentication message appeared on the bus, hence this attack reduces to a DoS attack.

C. Protocol Security = HMAC Security

We will now argue that the security of our protocol reduces to the security of the underlying HMAC algorithm. An adversary $\mathcal{A}$ is considered capable of breaking the protocol when he can forge authentication signatures for messages msg_i with probability $p > \epsilon$. Since signature content is entirely generated with HMAC, $\mathcal{A}$ can forge signatures sig_i' iif, given a message msg_i and a counter value ctr_i,

$$P\left(\text{sig}_i' = \text{sig}_i\right) > \epsilon.$$

Being able to forge signatures thus implies that $\mathcal{A}$ has knowledge of one of the following:

1) pre-shared key $\mathcal{K}p_i$
2) session key $\mathcal{K}s_i$
3) a key $\mathcal{K}s_i'$ for which

$$\text{HMAC}(\mathcal{K}s_i', i \parallel \text{ctr}_i \parallel \text{msg}_i) \bmod 2^{80} = \text{sig}_i.$$

Since invasive readings of node content is not possible in our adversarial model, option one is ruled out, since that would require invasive access to a node $\mathcal{N}$ storing $\mathcal{K}p_i$. The only way option two is possible then is if $\mathcal{A}$ manages to break HMAC. Finally, the third option requires $\mathcal{A}$ to find a collision on HMAC. Thus, the security of CANAuth depends entirely on the security of HMAC.

D. Tamper Resistance

The fact that a trusted node only increases its key establishment counter ctrA_i after a valid key establishment provides certain security benefits. If instead ctrA_i was increased after every unsuccessful key establishment attempt, $\mathcal{A}$ could easily mount a DoS attack by constantly raising errors, thereby rapidly increasing the counter to its maximum value and preventing any further key establishment. One could argue that this is not important, since our adversarial model ignores DoS attacks. However, a more important advantage of not increasing the counter ctrA_i is that it leaves the machine in which the CANAuth protocol is implemented in a non-functional state should $\mathcal{A}$ manage to establish a session key. Thus, this quality can work as a deterent against attacks: either $\mathcal{A}$ does not attempt any attacks and his machine stays functional or he has to completely break the protocol by finding $\mathcal{K}p_i$ to be able to keep his machine functional.

Assume $\mathcal{A}$ manages to find a valid tuple $\{\text{ctrA}_i, r_i, \mathcal{K}s_i\}$, but not $\mathcal{K}p_i$[5]. Due to the checks during the key establishment, ctrA_i needs to be higher than the last used authenticated counter value. Thus, if $\mathcal{A}$ uses the valid tuple, after the next machine reboot or when the message authentication counter reaches its maximum, the protocol will not accept any more valid key establishment messages from a trusted node, since the trusted node will transmit a key establishment counter value ctrA_i' equal to or lower than the ctrA_i value used by $\mathcal{A}$. Nodes will thus reject all (valid) messages from the trusted node. This means that if $\mathcal{A}$ ever uses such a valid tuple, the machine will be left in a non-functional state.

[5]Knowledge of $\mathcal{K}p_i$ would make this attack obsolete, since $\mathcal{A}$ can then gain full control over the messages $\mathcal{G}_i$ by either sending authenticated messages $\mathcal{G}_i$ himself or executing MitM attacks.

Note that this non-functionality can never happen during normal operation, since nodes only increase their key establishment counter value ctrA_i after a valid key establishment phase and only one node is allowed to set up session keys $\mathcal{K}s_i$ for $\mathcal{G}_i$, so counter values are always synchronized between verifying and signing nodes.

E. Limitations on On-line Attack Speed

Due to the nature of the CAN bus, message throughput on the bus is very modest at best. A table showing the time it takes to transmit messages is shown in Table I. These times are calculated assuming there are zero errors on the bus, an average number of stuffing bits[6] and include the necessary 3 bit interframe space after each message.

TABLE I

List of theoretical minimum transmission times for n messages of length l on an s baud CAN bus. An average number of stuffing bits was accounted for when calculating these times [1].

Baud rate s & message length l	Number of messages n 2^{16}	2^{32}	2^{64}
100 Kbps			
1 byte	39 s	29 d	3.4×10^8 y
4 bytes	54 s	2 mo	4.8×10^8 y
8 bytes	2 m	2 mo	6.7×10^8 y
500 Kbps			
1 byte	8 s	6 d	6.8×10^7 y
4 bytes	11 s	9 d	9.6×10^7 y
8 bytes	15 s	12 d	1.3×10^8 y
1 Mbps			
1 byte	4 s	3 d	3.4×10^7 y
4 bytes	6 s	5 d	4.8×10^7 y
8 bytes	8 s	6 d	6.7×10^7 y

As can be deduced from Table I, on-line attacks on the system, using honest nodes as oracles to verify a key guess, would be extremely slow. Furthermore, every wrong key guess leads to an error message on the bus, which in turn increases the *REC* of each node. If these nodes their *REC* exceeds 127, they will not signal errors anymore and can thus not be used as verification oracles. Thus, for every wrong key guess a valid message needs to appear on the bus, so that the honest nodes their *REC* does not exceed 127. So in reality, the time it would take to transmit a certain number of guesses would take at least twice as long as the times given in Table I.

VI. Conclusion

In this paper, we gave an overview of the problems associated with message authentication protocols on the CAN bus. Furthermore, we presented CANAuth, a simple, lightweight message authentication protocol based on HMAC for use on the CAN bus. Due to its backward compatibility, CANAuth-compatible nodes can work on a CAN bus without any modification to existing nodes. The security of the proposed protocol depends entirely on the security of the employed HMAC variant. Furthermore, due to the nature of the CAN bus and the design of CANAuth, adversaries executing on-line attacks against the protocol can only do this at very slow speeds and are severely hindered should they be able to establish a session key, further enhancing security.

One of the major drawbacks of the proposed protocol is that all nodes that must be able to verify messages $\mathcal{G}_i$ need to know the pre-shared key $\mathcal{K}p_i$. Thus, as future work, it would be interesting to research if there is any possibility of using some kind of asymmetric primitives whilst maintaining a sufficiently high level of security.

[6]There can only be a maximum of five bits of equal value in sequence on the CAN bus. Whenever such a series of five equal bits is encountered, the CAN controller inserts a stuffing bit of inverse value [1].

Acknowledgments

This work was supported in part the IAP Programme P6/26 BCRYPT of the Belgian State (Belgian Science Policy), by K.U.Leuven-BOF (OT/06/40), by FWO grant G.0300.07, and by the European Commission through the ICT programme under contract ICT-2007-216676 ECRYPT II.

References

[1] W. Voss, *A Comprehensible Guide to Controller Area Network*. Massachusetts, USA: Copperhill Media Corporation, 2005.

[2] 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 *IEEE Symposium on Security and Privacy*. IEEE Computer Society, 2010, pp. 447–462.

[3] S. Checkoway, D. McCoy, D. Anderson, B. Kantor, S. Savage, K. Koscher, A. Czeskis, F. Roesner, and T. Kohno, "Comprehensive Experimental Analysis of Automototive Attack Surfaces," in *Proceedings of the USENIX Security Symposium*, San Francisco, CA, August 2011.

[4] A. Perrig, R. Canetti, J. D. Tygar, and D. X. Song, "Efficient Authentication and Signing of Multicast Streams over Lossy Channels," in *IEEE Symposium on Security and Privacy*, 2000, pp. 56–73.

[5] A. Perrig, R. Canetti, D. X. Song, and J. D. Tygar, "Efficient and Secure Source Authentication for Multicast," in *NDSS*. The Internet Society, 2001.

[6] A. Perrig, R. Szewczyk, V. Wen, D. E. Culler, and J. D. Tygar, "SPINS: security protocols for sensor networks," in *MOBICOM*, 2001, pp. 189–199.

[7] R. Gennaro and P. Rohatgi, "How to sign digital streams," in *CRYPTO*, ser. Lecture Notes in Computer Science, B. S. K. Jr., Ed., vol. 1294. Springer, 1997, pp. 180–197.

[8] P. Rohatgi, "A compact and fast hybrid signature scheme for multicast packet authentication," in *ACM Conference on Computer and Communications Security*, 1999, pp. 93–100.

[9] H. Krawczyk, M. Bellare, and R. Canetti, "HMAC: Keyed-Hashing for Message Authentication," RFC 2104 (Informational), Internet Engineering Task Force, February 1997. [Online]. Available: http://www.ietf.org/rfc/rfc2104.txt

[10] M. Bellare, R. Canetti, and H. Krawczyk, "Keying Hash Functions for Message Authentication," in *CRYPTO*, ser. Lecture Notes in Computer Science, N. Koblitz, Ed., vol. 1109. Springer, 1996, pp. 1–15.

[11] T. Ziermann, S. Wildermann, and J. Teich, "CAN+: A new backward-compatible Controller Area Network (CAN) protocol with up to 16x higher data rates," in *DATE*. IEEE, 2009, pp. 1088–1093.