Formal Security Analysis of the MaCAN Protocol

Alessandro Bruni[1], Michal Sojka[2],
Flemming Nielson[1], and Hanne Riis Nielson[1]

[1] Technical University of Denmark
{albr,fnie,hrni}@dtu.dk
[2] Czech Technical University
sojkam1@fel.cvut.cz

Abstract. Embedded real-time network protocols such as the CAN bus cannot rely on off-the-shelf schemes for authentication, because of the bandwidth limitations imposed by the network. As a result, both academia and industry have proposed custom protocols that meet such constraints, with solutions that may be deemed insecure if considered out of context. MaCAN is one such compatible authentication protocol, proposed by Volkswagen Research and a strong candidate for being adopted by the automotive industry.

In this work we formally analyse MaCAN with ProVerif, an automated protocol verifier. Our formal analysis identifies two flaws in the original protocol: one creates unavailability concerns during key establishment, and the other allows re-using authenticated signals for different purposes. We propose and analyse a modification that improves its behaviour while fitting the constraints of CAN bus. Although the revised scheme improves the situation, it is still not completely secure. We argue that the modified protocol makes a good compromise between the desire to secure automotive systems and the limitations of CAN networks.

Keywords: protocol verification, embedded systems, Controller Area Network.

1 Introduction

The CAN Bus is a protocol for real-time broadcast communication introduced in 1983 by Bosch, and the current de facto standard for signal communication inside a modern vehicle. Its simplicity and long time of adoption have made it the most reasonable solution for building interoperable hardware in the automotive market. It is also a mandatory protocol for the OBD-II diagnostic interface in the United States and the similar EOBD standard for European countries. This also means that it is unlikely to be replaced by more powerful protocols in the foreseeable future (e.g. FlexRay [8]).

CAN is an unauthenticated broadcast protocol, therefore it offers no security-related features to system designers. As current vehicles get more interconnected

E. Albert and E. Sekerinski (Eds.): IFM 2014, LNCS 8739, pp. 241–255, 2014.

it is necessary to ensure that messages come from the right sources in order to secure car functionality that could result in safety hazards if compromised under an attack. Koscher et al. have shown [12,5] that current vehicles have practically no defence within their internal network. It is possible to perform an impressive array of attacks from interfaces like WiFi, Bluetooth and Cellular networks that are now available in modern vehicles.

In response to their study various groups have tried to secure the weak link in the chain, providing various flavours of authenticated CAN protocols: MaCAN [11], CANAuth [17], LiBrA-CAN [10] and Car2x [16]. Many of these schemes deviate from well-established communication protocols to meet the bandwidth and real-time constraints of the CAN network. Nevertheless, the authors of MaCAN and CANAuth, for example, only claim to guarantee certain security properties (e.g. authentication, freshness), without formally proving their correctness.

We analysed the MaCAN protocol as described in [11], with the assistance of the ProVerif protocol verifier [4]. Our analysis showed two flaws in the specification, one in the key distribution scheme, and another in signal authentication.

The first flaw allows the initiating principal to believe that a session has been established, while the other parties have not received a session key. Key distribution happens between three or more parties: an *initiator*, responsible for starting the procedure, the *key server*, responsible for delivering the session key, and one or more *responders*, which also need to obtain the session key.

The slightly asymmetric behaviour of the protocol allows an attacker to reuse the signature of the acknowledgement message sent by the initiator, in order to simulate an acknowledgement message coming from the responder, therefore completing authentication on one side. Furthermore, the attacker can manipulate the behaviour of the key server so that the responder never receives a request for authentication, and therefore the responder is never activated. This leads to an incomplete session establishment where one of the parties believes that it can communicate authenticated messages while the other will refuse such messages because it is not in possession of a valid session key.

Our proposed correction removes the asymmetry in the two phases of the protocol, and prevents the attack. We model our modification of the MaCAN protocol in ProVerif and discover another minor problem in the format of the acknowledgement message, that allows the attacker to successfully send acknowledgements with the signature of another principal in group sessions. Adding source information to the signature overcomes this annoyance, and allows us to prove the desired authentication property in the key establishment phase.

The second flaw allows repurposing an authenticated signal when a specific message format is used. An attacker can forge the signal number without this being detected, allowing, for example, the message with meaning “speed is 25” to be modified to “temperature is 25”.

Our correction modifies the signature so that the signal number is considered, preventing that particular attack form happening. However the nature of the protocol allows replays within the validity time frame of a message, which can

be rather long for its applications. Here we contribute with a discussion that clarifies which properties an application designer can expect from MaCAN, and needs to take into consideration when designing a system.

The paper proceeds as follows. Section 2 presents the applied π-calculus that we use in our models, Section 3 introduces the CAN bus protocol and its extensions, Section 4 describes the MaCAN protocol, Section 5 describes our formal analysis of the protocol, the discovered flaws and our proposed mitigations. We compare the result of our analysis with our implementation in Section 6 and present our conclusions in Section 7.

2 Modeling Protocols in ProVerif

ProVerif [4] is a protocol verifier that translates models in the applied π-calculus into a set of Prolog rules. Initially developed to verify secrecy properties [2], ProVerif uses a different resolution algorithm than Prolog's backward chaining to achieve better performance on the particular set of clauses that it generates.

ProVerif was later expanded to verify injective and non-injective agreements, by adding events to the applied π-calculus and extending the translation to verify non-injective and injective agreements, according to Lowe's [14] definitions. For a detailed presentation of the applied π-calculus with events and the analysis techniques used in this paper we refer to [3].

Figure 1 shows the language that we use in this paper. We have terms M, N which can be either variables, names, tuples or constructors applied to subterms. Patterns $\mathit{\Pi}$ are used in inputs and let bindings and are either variable binders, patterns on tuples or equality checks on terms. Processes P, Q are either the stuck process, the infinite replication of a process, the parallel composition which runs two processes in parallel, the restriction which binds a in P to a fresh name, input which applies pattern $\mathit{\Pi}$ to an input on channel M, output which outputs the term N on channel M, let which applies a rewrite rule of the form $g(M_1, \ldots, M_n) \to M$ — where $fv(M) \subseteq \bigcup_i fv(M_i)$ — and if it succeeds executes P after matching the result of the destruction to the pattern $\mathit{\Pi}$, otherwise executes Q, the if construct which checks equality between terms M and N and executes P if the two terms are equal, Q otherwise, and event, which signals an event in the execution of the process, marked with the term M.

3 CAN and Its Extensions

CAN [9] is a broadcast, prioritised, real-time protocol designed to work on a bus network topology of interconnected microcontrollers. At a high level, a CAN frame has two fields: *(i)* an identifier (CAN-ID) of either 11 or 29 bits, that is used for specifying which signal is being transmitted, and for arbitration purposes using Carrier Sense Multiple Access with Bitwise Arbitration (a multiple access scheme where lower ID values have priority on higher ID values and arbitration is resolved by sending the message IDs bit-by-bit), and *(ii)* a payload field of 1 to 8 bytes, that may contain information specific to the signal. Typical transmission

$$
\begin{array}{rll}
M, N ::= & x, y, z \mid a, b, c \mid (M_1, \ldots, M_n) \mid f(M_1, \ldots, M_n) & \text{terms} \\
\Pi ::= & x \mid (\Pi_1, \ldots, \Pi_n) \mid =M & \text{patterns} \\
P, Q ::= & 0 & \text{stuck process} \\
& \mid\ !P & \text{replication} \\
& \mid\ P|Q & \text{parallel composition} \\
& \mid\ (\nu\ a)\, P & \text{restriction} \\
& \mid\ M(\Pi).P & \text{input} \\
& \mid\ \overline{M}\langle N\rangle.P & \text{output} \\
& \mid\ \mathsf{let}\ \Pi = g(M_1, \ldots, M_n)\ \mathsf{in}\ P\ \mathsf{else}\ Q & \text{destructor application} \\
& \mid\ \mathsf{if}\ M = N\ \mathsf{then}\ P\ \mathsf{else}\ Q & \text{conditional} \\
& \mid\ \mathsf{event}(M).P & \text{event}
\end{array}
$$

Fig. 1. Applied π-calculus with events

speeds of the CAN bus are 125 KHz, 500 KHz and 1 MHz, and a single CAN frame occupies between 65 and 160 bits, depending on the length of the identifier field and of the payload. Other safety-related features of CAN include error detection using CRC codes and specific flags for signalling a failed transmission, and mechanisms for detecting and disabling malfunctioning devices.

Given the restricted size of CAN frames an authenticated network protocol must meet the tight space constraints. Splitting messages into more than one signal is rarely an acceptable solution, because most applications have real-time constraints and the bandwidth usage is approaching 80% [6].

To alleviate the payload limitation of CAN, two extensions have been proposed: CAN+ and CAN-FD. CAN+ [18] uses a clever encoding that allows to extend the payload by 15 times the original size, while being compatible with devices that use plain CAN, who see only the non-extended payload. This allows to use the extra bandwidth for authentication, as CANAuth [17] does. What hampers its adoption is the higher cost of required hardware, which has already been a compelling argument for industry to not switch to FlexRay.

CAN-FD [13] allows transmitting up to 64 bytes of payload by changing the data rate during payload transmission. This change is not transparent to standard CAN, but CAN and CAN-FD devices can coexist in the same network. CAN-FD is currently being integrated into ISO 11898, the standard defining CAN, so it is very likely to be adopted as a solution for extending the CAN payload. MaCAN has been designed to work on plain CAN networks, but it can be extended to use the extra payload offered by CAN-FD.

4 MaCAN

MaCAN [11] is an authenticated protocol specifically designed for the CAN bus. The authors argued that there was a need for a new authentication scheme that could fit into its small payload size. Other proposals such as CANAuth *rely* on

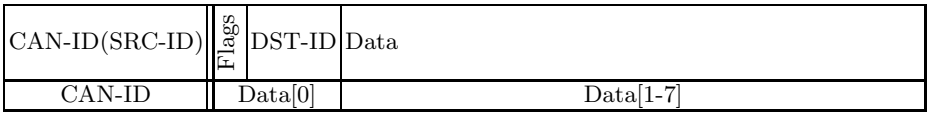

Fig. 2. The Crypt Frame

the extra bits offered by CAN+ and *require* also switching to different hardware. MaCAN instead authenticates messages in 4 bytes of payload, leaving the other 4 bytes available for signal information. The signature length can also be extended when using CAN-FD in order to increase robustness and to allow more than 4 bytes of information to be transmitted.

The designers of MaCAN discarded the use of more traditional challenge-response protocols during message authentication, because of the real-time requirements that need to be met in the applications. They also considered problematic the use of counter values to ensure message freshness, since there is not enough space to transmit the counter and a hash together with a message, and synchronisation issues make it impossible to keep track of the current counter value. Instead, they chose to authenticate messages using a timestamp to ensure their freshness, and to synchronise the ECUs using a time server. Timestamps need not to be sent with every frame as all ECUs have a local clock, hence MaCAN saves precious bandwidth for communication.

4.1 The Crypt Frame

As depicted in Figure 2, the crypt frame is a specific interpretation of the traditional CAN frame where both CAN-ID and payload fields are used to encode authentication details.

The CAN ID in the crypt frame encodes a 6 bit source ID, which indicates the source Electronic Control Unit (ECU). The first byte of data is used to send a 2 bit flag field and a 6 bit destination ID, which could indicate a specific ECU or a group of ECUs, in order to make the protocol fully directional. This leaves available the remaining 7 bytes for signals and signatures.

In the following sections we are going to present the protocol using Alice&Bob notation for the messages, which leaves the field lengths unspecified. The reader may refer to the original paper [11] for the specific frame formats.

4.2 Key Establishment

The key establishment procedure in MaCAN establishes a session key between an initiator (ECU_i) and a responder (ECU_j) by communicating to a key server (KS). Figure 3 represents the authentication process in Alice&Bob notation. Both ECU_i and ECU_j have their own pre-shared key $K_{x,ks}$ registered with the key server, which the key server uses for sending session keys.

To establish a session key, the initiator ECU_i sends a challenge C_i to KS, signalling the ID of the requested partner id_j (4.2.1). KS encrypts (senc) with

$$ECU_i \to KS : \mathsf{CH}, C_i, id_j \tag{4.2.1}$$
$$KS \to ECU_i : \mathsf{SK}, \mathsf{senc}((C_i, id_j, id_i, SK_{i,j}), K_{i,ks}) \tag{4.2.2}$$
$$KS \to ECU_j : \mathsf{RC} \tag{4.2.3}$$
$$ECU_i \to ECU_j : \mathsf{ACK}, group\_field, \mathsf{cmac}((T, id_j, group\_field), SK_{i,j}) \tag{4.2.4}$$
$$ECU_j \to KS : \mathsf{CH}, C_j, id_i \tag{4.2.5}$$
$$KS \to ECU_j : \mathsf{SK}, \mathsf{senc}((C_j, id_i, id_j, SK_{i,j}), K_{j,ks}) \tag{4.2.6}$$
$$ECU_j \to ECU_i : \mathsf{ACK}, group\_field, \mathsf{cmac}((T, id_j, group\_field), SK_{i,j}) \tag{4.2.7}$$
$$ECU_i \to ECU_j : \mathsf{ACK}, group\_field, \mathsf{cmac}((T, id_j, group\_field), SK_{i,j}) \tag{4.2.8}$$

Fig. 3. MaCAN key establishment procedure

$K_{i,ks}$ a fresh session key $SK_{i,j}$, together with the challenge C_i and the IDs of ECU_i and ECU_j (4.2.2). KS then sends a request for challenge (RC) to ECU_j, in order to activate it (4.2.3).

ECU_i after decrypting the session key $SK_{i,j}$ received from the key server, sends an acknowledgement (ACK) to ECU_j, signing it with $SK_{i,j}$ and the current timestamp T. The $group\_field$ is a bit vector that represents the knowledge of ECU_i about which devices are authenticated[1]. After sending this message, ECU_i considers itself authenticated, and awaits ECU_j to conclude the protocol.

ECU_j then sends its own challenge C_j to KS (4.2.5), to which the key server replies with the encrypted session key, C_j, and the IDs of the two principals (4.2.6). Finally ECU_j sends a signed acknowledgement message to ECU_i, signalling that it received the session key and updating $group\_field$ (4.2.7).

In case of group authentication, when an authenticated ECU receives a $group\_field$ that does not mark itself as authenticated, it sends an acknowledgement message to inform the other ECUs in the group of its presence (4.2.8).

4.3 Message Authentication

Automotive CAN applications are based on the concept of "signals" (e.g. current vehicle speed is a signal) that are exchanged between cooperating ECUs by periodically transmitting messages with signal values. In MaCAN it is possible to require ECUs to authenticate a specific signal upon request. Then next, each or each n-th signal message will be additionally sent in the authenticated format. Figure 4 shows how authenticated messages can be requested (4.3.1) and provided (4.3.2, 4.3.3). $Sig\#$ specifies the signal number that needs to be authenticated. $Prescaler$ specifies the signing behaviour, and is 0 to request the following message to be authenticated, 1 to request each following message to be

[1] MaCAN supports authentication of groups with more than two ECUs, but here we concentrate on the case where a session is established between two parties. Authenticating more than two ECUs requires the key server to interpret id_j as the ID of a group, which is statically defined, and to send requests for challenge to each ECU in the group.

$$ECU_i \to ECU_j : \mathsf{SIG\text{-}AUTH\text{-}REQ}, Sig\#, Prescaler, \mathsf{cmac}((T, id_i, id_j, Sig\#, Prescaler), SK_{i,j}) \quad (4.3.1)$$

$$ECU_j \to ECU_i : \mathsf{SIG\text{-}AUTH}, Sig\#, Signal, \mathsf{cmac}((T, id_i, id_j, Signal), SK_{i,j}) \quad (4.3.2)$$

$$ECU_j \to ECU_i : \mathsf{SIG\text{-}AUTH}, Signal, \mathsf{cmac}((T, id_i, id_j, Signal), SK_{i,j}) \quad (4.3.3)$$

$$ECU_i \to TS : \mathsf{CH}, C_i, fwd\_id = 0 \quad (4.4.1)$$

$$TS \to ECU_i : T, \mathsf{cmac}((C_i, T), SK_{ts,i}) \quad (4.4.2)$$

Fig. 4. MaCAN signal authentication (4.3.1-3) and time requests (4.4.1-2)

authenticated, and any $n > 1$ to request each n-th message to be authenticated. The CMAC [7] signature uses the current timestamp T, the initiator and the responder IDs, to authenticate the request.

The responder to the authentication request replies with either (4.3.3) or (4.3.2), depending on whether the signal value ($Signal$) fits in 32 bits or not.

4.4 Serving Time

All signatures include a timestamp that is not sent in clear-text over the channel, and thus the communicating devices need to be synchronised, otherwise there is a risk that authenticated messages might not be recognised as valid. To mitigate unavailability concerns due to clock synchronisation, MaCAN introduces an authenticated time-serving protocol, shown in Figure 4.

Normally the current timestamp is broadcast periodically in an unauthenticated form. When ECU_i detects too big a mismatch between the internal clock and the received timestamp, it may send a request for an authentic time value from the time server. This is done by sending a challenge C_i to the time server (4.4.1), who will then reply with the last broadcasted timestamp T, signed using the challenge and a session key $SK_{ts,i}$ shared by the time server and the ECU.

5 Formal Analysis

5.1 Key Establishment

Figure 5 shows our model of the MaCAN authentication procedure in the applied π-calculus. All communication happens on a broadcast channel c, while we use a private channel psk for the key server to store the long term keys of the ECUs. The process KS represents the key server, ECU_i is the initiator process and ECU_j is the responder process.

Due to the abstractions introduced by ProVerif, we have to change some important aspects of the protocol in order to obtain a precise analysis. First and foremost, we remove timestamps from signatures, because ProVerif abstracts away the concept of state in its translation of processes to Horn clauses. Then

$$
\begin{aligned}
KS \triangleq\ & c(i, =\mathsf{CH}, =ks, c_i, j).psk(=i, k_i).(\nu\ sk_{ij})\ \mathsf{event}(sessk_i(i, j, c_i, sk_{ij})).\\
& \overline{c}\langle ks, \mathsf{SK}, i, \mathsf{senc}((c_i, j, i, sk_{ij}), k_i)\rangle.\overline{c}\langle ks, \mathsf{RC}, j\rangle.\\
& c(=j, =\mathsf{CH}, =ks, c_j, =i).psk(=j, k_j).\mathsf{event}(sessk_j(j, i, c_i, sk_{ij})).\\
& \overline{c}\langle ks, \mathsf{SK}, j, \mathsf{senc}((c_j, i, j, sk_{ij}), k_j)\rangle.c(=sk_{ij}).\overline{c}\langle error\rangle.0
\end{aligned}
$$

$$
\begin{aligned}
ECU_i \triangleq\ & (\nu\ c_i)\ \mathsf{event}(authStart_i(i, j, c_i)).\overline{c}\langle i, \mathsf{CH}, ks, c_i, j\rangle.\\
& c(=ks, =\mathsf{SK}, =i, resp).\mathsf{let}\ (=c_i, =j, =i, sk_{ij}) = \mathsf{sdec}(resp, k_i)\ \mathsf{in}\\
& \mathsf{event}(authAck_i(i, j, c_i, sk_{ij})).\overline{c}\langle i, \mathsf{ACK}, j, sign((j, AK), sk_{ij})\rangle.\\
& c(=j, =\mathsf{ACK}, =i, =sign((j, \mathsf{ACK}), sk_{ij})).\mathsf{event}(authEnd_i(i, j, c_i, sk_{ij})).0
\end{aligned}
$$

$$
\begin{aligned}
ECU_j \triangleq\ & c(=ks, =\mathsf{RC}, =j)).c(i, =\mathsf{ACK}, =j, ack).(\nu\ c_j)\\
& \mathsf{event}(authStart_j(j, i, c_j)).\overline{c}\langle j, \mathsf{CH}, ks, c_j, i\rangle.c(=ks, =\mathsf{SK}, =j, resp).\\
& \mathsf{let}\ (=c_j, =i, =j, sk_{ij}) = \mathsf{sdec}(resp, k_j)\ \mathsf{in}\\
& \mathsf{if}\ ack = sign((j, \mathsf{ACK}), sk_{ij})\ \mathsf{then}\ \mathsf{event}(authAck_j(j, i, c_j, sk_{ij})).\\
& \overline{c}(j, AK, i, sign((j, \mathsf{ACK}), sk_{ij})).\mathsf{event}(authEnd_j(j, i, c_j, sk_{ij})).0
\end{aligned}
$$

Fig. 5. MaCAN key establishment process in the applied π-calculus

we treat $group\_field$ as a name instead of a bit vector, in order to simplify the model. Finally we encode long encrypted messages that would be split into multiple frame as a single message. We take these changes into consideration when we interpret the results of our analysis and we argue to which extent they introduce overapproximations.

Current MaCAN configurations have clock rates of 1 second, so it is safe to assume that timestamps can be treated as constants, since the key establishment procedure can complete within a single clock tick. Note that it is undesirable to have high clock rates due to the following constraint: the receiving end of an authenticated signal needs to check a signature against all valid timestamps within the possible reception window of the message. Therefore, increasing the clock rate also requires more computation on the receiving end, which in turn increases the worst case response time for a signal transmission. The length of the reception window for a message can be obtained with schedulability analysis [6] and depends on the number of higher priority messages that can delay the transmission of the message in question.

In Figure 5, KS represent the key server process. It waits on the public channel for a challenge c_i to establish a session between ECU_i and ECU_j, retrieves k_i from its database, produces a fresh session key sk_{ij}, outputs the encoding of the session and sends a request for challenge to ECU_j. It then waits for a challenge c_j from ECU_j, retrieves its key k_j, and encodes the session key sk_{ij} in a message for ECU_j that includes the challenge c_j. Finally it waits for the session key to be sent in clear text on the channel to signal an error. If an error is not reachable then the secrecy of sk_{ij} is guaranteed.

$$authStart_i(id_i, id_j, C_i) \tag{5.1.1}$$
$$ECU_i \rightsquigarrow KS : \mathsf{CH}, C_i, id_j \tag{5.1.2}$$
$$\mathrm{M}[ECU_j] \rightarrow KS : \mathsf{CH}, a, id_i \tag{5.1.3}$$
$$sessk_i(id_j, id_i, a, SK_{i,j}) \tag{5.1.4}$$
$$KS \rightarrow ECU_j : \mathsf{SK}, \mathsf{senc}((a, id_i, id_j, SK_{i,j}), K_{ks,j}) \tag{5.1.5}$$
$$KS \rightsquigarrow ECU_i : \mathsf{RC} \tag{5.1.6}$$
$$\mathrm{M}[ECU_i] \rightarrow KS : \mathsf{CH}, C_i, id_j \tag{5.1.7}$$
$$sessk_j(id_i, id_j, C_i, SK_{i,j}) \tag{5.1.8}$$
$$KS \rightarrow ECU_i : \mathsf{SK}, \mathsf{senc}((C_i, id_j, id_i, SK_{i,j}), K_{ks,i}) \tag{5.1.9}$$
$$authAck_i(id_i, id_j, SK_{i,j}) \tag{5.1.10}$$
$$ECU_i \rightarrow ECU_j : \mathsf{ACK}, \mathsf{cmac}((T, id_j, group\_field), SK_{i,j}) \tag{5.1.11}$$
$$M[ECU_j] \rightarrow ECU_i : \mathsf{ACK}, \mathsf{cmac}((T, id_j, group\_field), SK_{i,j}) \tag{5.1.12}$$
$$authEnd_i(id_i, id_j, SK_{i,j}) \tag{5.1.13}$$

Fig. 6. Attack trace

ECU_i creates a new challenge c_i sends the challenge to the key server, waits for the response of the key server and decodes the message to retrieve the session key sk_{ij}. ECU_i then sends an acknowledgement to ECU_j signed with sk_{ij}, and waits for a similar acknowledgement from ECU_j to conclude the key establishment procedure.

ECU_j waits for a request for challenge from the key server, reads the acknowledgement from ECU_i, sends its challenge to the key server, receives the session key sk_{ij}, verifies the validity of the acknowledgement from the other party and finally sends its own acknowledgement, concluding its part of the procedure.

Analysis results. We analysed the following five properties for key establishment:

- *(i)* the secrecy of long term keys k_i, k_j,
- *(ii)* the secrecy of session keys ks_{ij},
- *(iii)* the agreement between the events $authStart_i(i, j, c_i)$, $sessk_i(i, j, c_i, sk_{ij})$, $authAck_i(i, j, c_i, sk_{ij})$, $authEnd_i(i, j, c_i, sk_{ij})$, and
- *(iv)* the agreement between the events $authStart_j(j, i, c_j)$, $sessk_j(j, i, c_j, sk_{ij})$, $authAck_j(j, i, c_j, sk_{ij})$, $authEnd_j(j, i, c_j, sk_{ij})$.

Using ProVerif, we were able to verify the secrecy properties *(i,ii)*, but we found a counterexample for the event correspondence *(iii)*, where an attacker can run the protocol in such a way that ECU_i receives the proper session key from message (4.2.6) instead of (4.2.2), leaving the ECU_j unauthenticated. The correspondence *(iv)* for ECU_j is proven, therefore it can only authenticate as intended by the protocol.

Figure 6 shows our reconstruction of the attack trace produced by ProVerif for the query of events related to ECU_i (property *iii*), thereby providing feedback

$$ECU_i \to KS : \mathsf{CH}, C_i, id_j \tag{5.1.14}$$
$$KS \to ECU_i : \mathsf{SK}, \mathsf{senc}((C_i, id_j, id_i, SK_{i,j}), K_{i,ks}) \tag{5.1.15}$$
$$KS \to ECU_j : \mathsf{RC} \tag{5.1.16}$$
$$ECU_i \to ECU_j : \mathsf{ACK}, group\_field, \mathsf{cmac}((T, id_i, id_j, group\_field), SK_{i,j}) \tag{5.1.17}$$
$$ECU_j \to KS : \mathsf{CH}, C_j, id_i \tag{5.1.18}$$
$$KS \to ECU_j : \mathsf{SK}, \mathsf{senc}((C_j, id_i, id_j, SK_{i,j}), K_{j,ks}) \tag{5.1.19}$$
$$ECU_j \to ECU_i : \mathsf{ACK}, group\_field, \mathsf{cmac}((T, id_j, id_i, group\_field), SK_{i,j}) \tag{5.1.20}$$

Fig. 7. Modified MaCAN key establishment procedure

to the protocol designer about how to amend the protocol. In this trace "$\rightsquigarrow$" represents a message deleted by the attacker (this can be achieved by jamming the signal at the proper time or by making one of the participating nodes or an involved CAN gateway unavailable) and M[x] represent the malicious agent impersonating x (it can be done by sending a message with the proper CAN-ID).

This attack relies on the possibility to remove messages from the channel. The attacker learns the current challenge and the destination ID (5.1.2), while suppressing the message. It then impersonates ECU_j and starts sending a random challenge (5.1.3), initiating the communication with the key server in the opposite direction. The key server then sends a legitimate message to ECU_j (5.1.5), who will ignore it as it did not request a session key. Then the key server sends a request for challenge to ECU_i (5.1.6), which may be suppressed by the attacker. The attacker remembers the previous challenge from ECU_i and replays it on the key server (5.1.7), receiving the session key encrypted for ECU_i in return (5.1.9). Finally ECU_i sends its acknowledgement message (5.1.11), and since the form of the two acknowledgement messages is the same for ECU_i (4.2.4) and ECU_j (4.2.7), the attacker can impersonate ECU_j and send back the same signature (5.1.12) so that ECU_i believes that also ECU_j is authenticated.

Corrected model. We propose a correction of the model where the asymmetries that cause the improper authentication behaviour are removed. Figure 7 shows the corrected procedure.

To guarantee the agreement property we modify the form of the acknowledgement message. The CMAC signature is now using the current timestamp, the source and the destination of the message as content. Because CMAC is a hashed signature, adding more parameters to the function does not affect the final payload size, therefore the modified protocol still fits the space constraints of CAN. The two acknowledgement messages (5.1.17) and (5.1.20) are now symmetrical. We added the source information on the signed hashes, as well as the destination. This allows not only to prove the necessary correspondence for two-party sessions, but in case of group sessions it removes the chance for an intruder to reuse an acknowledgement message of another principal.

$$
\begin{aligned}
KS \triangleq\ & c(i, =\mathsf{CH}, =ks, c_i, j).psk(=i, k_i).(\nu\ sk_{ij})\,\mathsf{event}(sessk_i(i, j, c_i, sk_{ij})).\\
& \overline{c}\langle ks, \mathsf{SK}, i, \mathsf{senc}((c_i, j, i, sk_{ij}), k_i)\rangle.\overline{c}\langle ks, \mathsf{RC}, j, i\rangle.\\
& c(=j, =\mathsf{CH}, =ks, c_j, =i).psk(=j, k_j).\mathsf{event}(sessk_j(j, i, c_i, sk_{ij})).\\
& \overline{c}\langle ks, \mathsf{SK}, j, \mathsf{senc}((c_j, i, j, sk_{ij}), k_j)\rangle.c(=sk_{ij}).\overline{c}\langle error\rangle.0
\end{aligned}
$$

$$
\begin{aligned}
ECU_i \triangleq\ & (\nu\ c_i)\,\mathsf{event}(authStart_i(i, j, c_i)).\overline{c}\langle i, \mathsf{CH}, ks, c_i, j\rangle.\\
& c(=ks, =\mathsf{SK}, =i, resp).\mathsf{let}\ (=c_i, =j, =i, sk_{ij}) = \mathsf{sdec}(resp, k_i)\ \mathsf{in}\\
& \mathsf{event}(authAck_i(i, j, c_i, sk_{ij})).\overline{c}\langle i, \mathsf{ACK}, j, sign((i, j, AK), sk_{ij})\rangle.\\
& c(=j, =\mathsf{ACK}, =i, =sign((j, i, \mathsf{ACK}), sk_{ij})).\mathsf{event}(authEnd_i(i, j, c_i, sk_{ij})).0
\end{aligned}
$$

$$
\begin{aligned}
ECU_j \triangleq\ & c(=ks, =\mathsf{RC}, =j, i).(\nu\ c_j)\,\mathsf{event}(authStart_j(j, i, c_j)).\\
& \overline{c}\langle j, \mathsf{CH}, ks, c_j, i\rangle.c(i, =\mathsf{ACK}, =j, ack).c(=ks, =\mathsf{SK}, =j, resp).\\
& \mathsf{let}\ (=c_j, =i, =j, sk_{ij}) = \mathsf{sdec}(resp, k_j)\ \mathsf{in}\\
& \mathsf{if}\ ack = sign((i, j, \mathsf{ACK}), sk_{ij})\ \mathsf{then}\ \mathsf{event}(authAck_j(j, i, c_j, sk_{ij})).\\
& \overline{c}(j, AK, i, sign((j, i, \mathsf{ACK}), sk_{ij})).\mathsf{event}(authEnd_j(j, i, c_j, sk_{ij})).0
\end{aligned}
$$

Fig. 8. MaCAN key establishment process in the applied π-calculus

Figure 8 shows the corrected model in the applied π-calculus, where we applied the modified behaviour for the three processes. The properties *(i–iv)* that we defined in Section 5.1 have all been proved in this model.

5.2 MaCAN Message Authentication

During a session, authenticated parties can send authenticated signals. As described in Section 4.3 the transmission of an authenticated signal needs to follow a specific request (4.3.1). Depending on whether the authenticated signal fits in 32 bits — that is half of the available CAN payload size — the responding ECU uses either message format (4.3.3) or (4.3.2).

Figure 9 shows two communicating processes that exchange authenticated messages according to message format (4.3.2). ECU_i requests an authenticated signal with the first output according to (4.3.1). Then it keeps waiting for an authenticated signal and checks whether the signature corresponds to its own computation of it, marking with an *accept* the acceptance of an authenticated signal. On the other side ECU_j receives a request for authentication, checks its signature and starts sending signals, marking with a *send* event the transmission of a fresh signal.

The original paper [11] is not clear about whether the CMAC signature includes the signal number. The process in Figure 9 does not include the signal number as part of the signature for signals. Thus the correspondence between $send(sig\#, signal)$ and $accept(sig\#, signal)$ is not verified. An attacker can read

$$
\begin{aligned}
ECU_i \triangleq\ & \overline{c}\langle i, \mathsf{SIG\text{-}AUTH\text{-}REQ}, j, sig\#, n0, \mathsf{cmac}((i, j, sig\#, n0), sk_{ij})\rangle. \\
& !(c(=j, =\mathsf{SIG\text{-}AUTH}, =i, sig\#, signal, x_{sig}). \\
& \quad \mathsf{if}\ x_{sig} = \mathsf{cmac}((i, j, signal), sk_{ij})\ \mathsf{then} \\
& \quad \mathsf{event}(accept(sig\#, signal)).0)
\end{aligned}
$$

$$
\begin{aligned}
ECU_j \triangleq\ & c(i, =\mathsf{SIG\text{-}AUTH\text{-}REQ}, =j, sig\#, prescaler, x_s). \\
& \mathsf{if}\ x_s = \mathsf{cmac}((i, j, sig\#, prescaler), k_{ij})\ \mathsf{then} \\
& !((\nu\ signal)\, \mathsf{event}(send(sig\#, signal)). \\
& \quad \overline{c}\langle j, \mathsf{SIG\text{-}AUTH}, i, sig\#, signal, \mathsf{cmac}((i, j, signal), sk_{ij})\rangle.0)
\end{aligned}
$$

Fig. 9. MaCAN message authentication processes in the applied π-calculus

a signed signal with a $sig\#$ value re-transmit the signal with a different $sig\#$ if multiple $sig\#$ have been requested within the same session.

A simple solution to this problem is to add $sig\#$ as part of the signature. With the modified process, which we omit for sake of brevity, we are able to verify the agreement between the events $send(sig\#, signal)$ and $accept(sig\#, signal)$.

Still we fail to verify an injective agreement between the two events. Given our specification it is possible to accept twice the same message, and this could constitute a freshness violation. Our abstraction removes all timestamps, as the modelling technology cannot efficiently deal with them, and we previously argued that they can be ignored and treated as constant within their validity window.

As current configurations have clock rates of one second, this constitutes a potentially serious flaw in the protocol. Imagine MaCAN authenticating messages for the brake control unit of a vehicle. In case of emergency braking at high speed, the driver might be tempted to go all the way down with the foot on the brake pedal, activating the ABS. The ABS control unit works by sending messages to the brake control unit, releasing the brakes at a fast interval, so that the wheels don't slide on the ground, reducing their grip. In this example an attacker could wait for a "release message" from the ABS control unit, and replay it for its whole validity, therefore effectively disabling the brake for an entire second in a dangerous situation.

Given the restrictions imposed by the CAN bus, we believe that MaCAN constitutes a good enough solution for authenticating signals. A better level of security can be achieved by incrementing the clock rate. This would reduce the time window available for replaying messages, and therefore reduce the potential effect of such replay. In case of fast control loops — where a specific signal needs to be sent every 50 ms, for example — a solution that completely prevents replay attacks would synchronise the clock with the message rate, and refuse any message signed with a timestamp that has been previously used. Specific care would then be required for synchronising the clock between the communicating devices, and to avoid any unavailability issues due to improper synchronisation.

In our work we also analysed CANAuth, an alternative proposal for an authenticated protocol on top of CAN, which appears to be immune from this kind of replay attacks. CANAuth uses counters as part of the signature for a message, and a receiving ECU accepts a message only if the counter is higher than any other counter value previously observed. This mechanism works, but relies on the extra bandwidth provided by CAN+ for transmitting both the signature and the current counter value. To our knowledge, MaCAN is the only protocol with authentication that is able to fit into the constrained frame of CAN bus.

6 Discussion

Abstraction gap. We developed our models of the MaCAN protocol in the applied π-calculus of ProVerif. We had to change some aspects of MaCAN, as described in Section 5, to be able to analyse it. One of such aspects is the use of timestamps to ensure freshness of messages. We were unsuccessful in modeling timestamps in ProVerif, as the tool abstracts away state information, and therefore we were not able to express freshness of timestamp values.

Other analysers such as StatVerif [1] have a global synchronised state, but in order to represent potentially infinite timestamps one needs more powerful abstractions that avoid exploring an infinite state-space. Explicitly inserting a fresh timestamp into a list and checking whether the current timestamp is in the list, for example, would generate terms of continuously increasing size, hanging the engine. The same behaviour we encountered in ProVerif, not surprisingly, as they share the same resolution engine.

A possible solution is the one offered by AIF [15], which abstracts values into the sets to which they belong: for example timestamps could be abstracted into current and expired ones by using two sets. We are currently working on implementing a similar abstraction on top of the applied π-calculus, in order to model more directly security protocols as communicating processes, reducing the distance between the model used for verification and the concrete program.

Implementation. We compared the result of our analysis with our implementation of MaCAN, developed independently from the analysis, which is available under an opensource license on Github[2]. We implemented the attacks to the key establishment and message authentication procedure, putting the attacker in control of a gateway as shown in Figure 10, which was demonstrated to be possible in practice by Checkoway et al. [5].

The attack on key establishment was possible only after aligning the implementation to the specification contained in the paper, as the necessary acknowledgement was already corrected in our implementation. We cannot trace back, however, whether this correction was due to explicit considerations by our skilled engineers, or it happened by chance by misinterpreting the flawed specification.

Our implementation also accepted authenticated acknowledgement messages replayed by the attacker impersonating another device in group authentication,

[2] `https://github.com/CTU-IIG/macan`

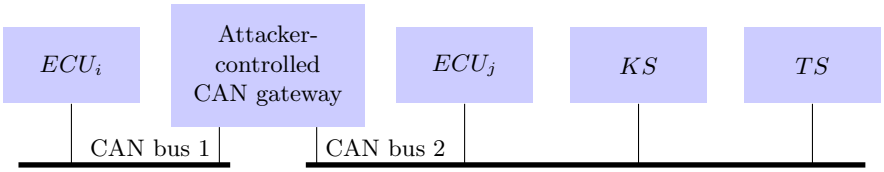

Fig. 10. Experimental setup

however with no practical consequences. We believe that this is a dangerous mistake to leave in a reference paper, which could lead to flawed implementations if left undetected. We could confirm the attack that allowed forging authenticated signals by changing signal number was present in our implementation. With it an attacker could forge potentially dangerous authenticated messages from legitimate ones. Comparing the models with the implementation also helped us to reveal some minor bugs that were introduced when coding it, and would have probably not been revealed by simple testing.

7 Conclusions

In this work we analysed MaCAN using the ProVerif protocol verifier, found a flaw in the key establishment procedure, experimentally verified the presence of an attack in our implementation, and proposed a modified version of the protocol that is immune from the problems that we discovered.

Resource constrained networks such as the CAN bus put a strong limit on the design of an authenticated protocol. The designers of MaCAN had to rely of custom schemes when designing its procedures, as previous literature did not consider such extreme bounds in terms of bandwidth as 8 bytes of payload per message. We contribute to the protocol with a formal and experimental analysis of its procedures and propose two changes that improve its behaviour.

During our analysis we also encountered some limitations in expressing the particular features of MaCAN with the languages and tools of our choice. We are currently working on an extension of the applied π-calculus that allows us to better model protocols with timestamps and counters.

Finally, protocols like MaCAN rely on relatively weak cryptography, so we would like to extend our analysis to cover possible attacks in the computational model, and be able to precisely evaluate the level of security of MaCAN.

Acknowledgments. This work is supported by the European project SESAMO. The authors would like to thank Roberto Vigo for valuable discussion on the models and the anonymous reviewers for helpful comments.

References

1. Arapinis, M., Ritter, E., Ryan, M.D.: StatVerif: Verification of Stateful Processes. In: 2011 IEEE 24th Computer Security Foundations, pp. 33–47 (2011)
2. Blanchet, B.: An efficient cryptographic protocol verifier based on prolog rules. In: Proceedings of the 14th IEEE workshop on Computer Security Foundations, pp. 82–96. IEEE Computer Society (2001)
3. Blanchet, B.: Automatic verification of correspondences for security protocols. Journal of Computer Security 17(4), 363–434 (2009)
4. Blanchet, B., Smyth, B.: Proverif 1.88: Automatic cryptographic protocol verifier, user manual and tutorial (2013)
5. Checkoway, S., McCoy, D., Kantor, B., Anderson, D., Shacham, H., Savage, S., Koscher, K., Czeskis, A., Roesner, F., Kohno, T.: Comprehensive experimental analyses of automotive attack surfaces. In: Proceedings of the 20th USENIX Conference on Security (2011)
6. Davis, R.I., Burns, A., Bril, R.J., Lukkien, J.J.: Controller area network (CAN) schedulability analysis: Refuted, revisited and revised. Real-Time Systems 35(3), 239–272 (2007)
7. Dworkin, M.J.: SP 800-38B. recommendation for block cipher modes of operation: the CMAC mode for authentication (2005)
8. FlexRay Consortium, et al.: FlexRay communications system-protocol specification. Version, 2(1):198–207 (2005)
9. Robert Bosch GmbH. Road vehicles – Controller area network (CAN) – Part 1: Data link layer and physical signalling (1991)
10. Groza, B., Murvay, S., Van Herrewege, A., Verbauwhede, I.: LiBrA-CAN: a Lightweight Broadcast Authentication protocol for Controller Area Networks (2012)
11. Hartkopp, O., Reuber, C., Schilling, R.: MaCAN - message authenticated CAN. In: Proceedings of the 10th Escar Conference on Embedded Security in Cars (2012)
12. Koscher, K., Czeskis, A., Roesner, F., Patel, S., Kohno, T., Checkoway, S., McCoy, D., Kantor, B., Anderson, D., Shacham, H., Savage, S.: Experimental Security Analysis of a Modern Automobile. In: 2010 IEEE Symposium on Security and Privacy, pp. 447–462 (2010)
13. Hartwich, F.: CAN with Flexible Data-Rate (2005)
14. Lowe, G.: A hierarchy of authentication specifications. In: Proceedings of the 10th Computer Security Foundations Workshop, pp. 31–43. IEEE (1997)
15. Mödersheim, S.A.: Abstraction by set-membership: verifying security protocols and web services with databases. In: Proceedings of the 17th ACM Conference on Computer and Communications Security, pp. 351–360. ACM (2010)
16. Schweppe, H., Roudier, Y., Weyl, B., Apvrille, L., Scheuermann, D.: Car2x communication: securing the last meter-a cost-effective approach for ensuring trust in car2x applications using in-vehicle symmetric cryptography. In: Vehicular Technology Conference (VTC Fall), pp. 1–5. IEEE (2011)
17. Van Herrewege, A., Singelee, D., Verbauwhede, I.: CANAuth — a simple, backward compatible broadcast authentication protocol for CAN bus. In: ECRYPT Workshop on Lightweight Cryptography 2011 (2011)
18. Ziermann, T., Wildermann, S., Teich, J.: CAN+: A new backward-compatible Controller Area Network (CAN) protocol with up to 16× higher data rates. In: Design, Automation & Test in Europe Conference & Exhibition, pp. 1088–1093. IEEE (2009)