Plug-and-secure communication for CAN

Security is a topic of rapidly increasing importance in automotive as well as industrial applications. With Bosch's novel approach, symmetric cryptographic keys between different nodes in a CAN network can simply be established and refreshed.

The recent trend towards ubiquitously connected systems – be it cars, factories, or buildings – does not only come with numerous opportunities and benefits, but imposes also new security threats with a potentially huge impact. If everything is interconnected and APIs are introduced in order to facilitate innovative services and applications, the attack surface for malicious manipulations and intrusions is increased significantly. Without proper countermeasures, hackers may remotely control cars, eavesdrop on confidential production data, or manipulate a building automation system.

This threat is reflected by various prominent attacks that have been performed and published recently. In "Remote exploitation of an unaltered passenger vehicle" [1], for example, the authors describe how they managed to remotely inject messages to the CAN network of a car and thus affect important physical systems, such as steering or braking. The hackers connected to the vehicle via a mobile network. This led to a recall of about 1,4 million cars and fueled legislative initiatives to mandate car manufacturers to support reasonable measures to protect cars against hacking attacks [2]. More security leaks and attacks on cars and other vehicles have been reported reported [3]-[5]. Remote attacks may easily scale and hackers do not need physical access to the system under attack. For example, imagine a scenario with thousands of cars remotely hijacked. Hackers could precipitate a breakdown of the whole traffic infrastructure of a city or country by manipulating all cars in a coordinated manner. Clearly, this could not only lead to tremendous physical damage, but also to a significant impact on the whole economy and society. Therefore, the support of appropriate security mechanisms represents a crucial prerequisite for the success and acceptance of any connected system.

In automotive networks, a secured CAN communication represents a particularly important building block of security concepts, since a compromised CAN network can have a direct impact on people's safety. When CAN was introduced in the 1980s, security was not a crucial topic yet, since systems were closed and isolated. With the increased openness, however, this changes fundamentally. While in principle suitable concepts and algorithms are available [6], [7], they are not used yet because other challenges still remain. This includes proper standardization across different manufacturers and suppliers, but also efficient approaches for establishing, refreshing, and managing the cryptographic keys that are required for the involved cryptographic schemes.

We therefore propose a novel approach addressing the latter aspect: this approach is able to establish and refresh symmetric cryptographic keys between two nodes in a CAN network in a plug-and-play manner. To this end, special properties of the CAN physical layer are exploited. It can be implemented with no or only minor extensions to CAN controllers and it is particularly suited to enhance the security against remote (and thus scalable) attacks. The generation of group keys between a group of nodes becomes possible if the approach is integrated into a suitable protocol flow.

System and attacker model

In the following, we consider a setup as depicted in Figure 1. Two devices (Alice and Bob) are connected to the same CAN network segment and want to establish a pair of symmetric cryptographic keys. Afterwards, they may use these keys to encrypt and/or authenticate any messages exchanged between them. In addition, there is also a potential attacker (Eve) connected to the same bus segment, which tries to determine or influence the keys to be established between Alice and Bob. We make the following assumptions on Alice, Bob, and Eve:

- All nodes have a similar setup, made up of a CAN transceiver, a suitable CAN controller, as well as a microprocessor running the application software,
- Eve is the victim of a remote attack in the sense that the original software running on that node has been replaced by a modified software,
- Eve can eavesdrop on all messages exchanged on the CAN network. She may also inject arbitrary single bits on the bus by bypassing the CAN controller and directly accessing the CAN transceiver from the malicious software running on the device.

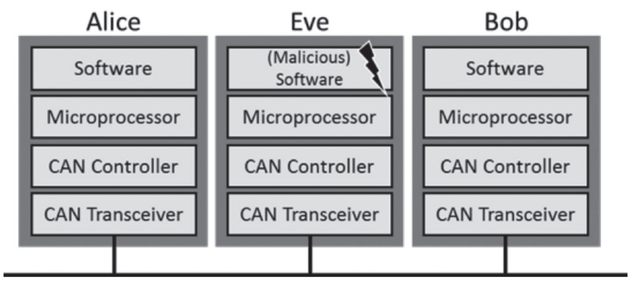


Figure 1: Considered System Model (Photo: Bosch)

It is important to make sure that if one device has been successfully attacked (here: Eve), the impact on the overall system is kept to a minimum. In the attack on a regular car reported in [1], the head unit was successfully compromised first. With proper security mechanisms in place, it would be hardly possible for the head unit to control safety-relevant functions like steering or braking by injecting CAN messages. However, any security mechanism is only secure ▷

as long as the involved cryptographic keys of the legitimate nodes (here: Alice and Bob) remain secret. Therefore, Eve must not be able to determine and/or influence these keys.

Review: Security for CAN networks

The protection of the integrity of a message and the assurance of the authenticity of its sender should be among the top security goals in CAN-based networks. Unauthorized manipulations have to be prevented or should at least be detectable. Encryption, in contrast, makes it harder for an attacker to learn the current system state or becomes necessary for delivering critical software updates.

In principle, all these things could be realized in exactly the same way as in the conventional IT world, but for optimal performance the constraints of CAN-based networks must be taken into account. This includes the limited data rate and message sizes, as well as the limited computational power and memory of CAN devices. Therefore, [6] and [7] propose several security mechanisms specifically optimized for CAN. Symmetric cryptographic schemes turn out to be the basis for most of the proposed schemes due to their limited computational complexity and bandwidth requirements. The use of symmetric cryptography requires the availability of symmetric keys at the involved nodes and the distribution and establishment of these keys represents a challenge. Possible options include a manual distribution of keys, e.g., at the end of a production line. This involves a considerable organizational effort and is made invalid if one or several devices were compromised before being integrated into the network. Besides, an automated refreshment of keys cannot be realized this way. An alternative approach is the use of key establishment schemes based on asymmetric cryptography, such as the Diffie-Hellman key exchange protocol [8], [9]. Drawbacks of this approach are the high computational complexity as well as the large amount of data that has to be exchanged in order to set up a secure symmetric key. Moreover, the security of the Diffie-Hellman key exchange relies only on the difficulty to efficiently solve the discrete logarithm problem on finite fields or elliptic curves using state-of-the-art methods. Therefore, once an efficient algorithm for solving these problems has been found, the approach may suddenly become insecure.

We therefore propose a novel approach for establishing symmetric cryptographic keys between two nodes, whose security does not rely on hard mathematical problems, but rather on physical properties of the CAN network. Furthermore, it has a low complexity, low bandwidth requirements, and can be implemented in existing systems. Finally, it may also be used for refreshing established keys.

CAN-based key establishment

The basic idea of our approach is that Alice and Bob agree on a shared secret or key by means of a public discussion using CAN messages. In particular, both nodes simultaneously transmit CAN frames, so that Eve can only see the superposition of both messages. However, since Alice and Bob themselves know what they have transmitted and since they can see the superposition of both messages as well, they may conclude what the other peer has ▷

transmitted and thus establish a shared key that is unknown to Eve.

For the realization, we rely on the property of the CAN network that bit ‘0’ is dominant and bit ‘1’ is recessive, which represents also the basis for the classical bus arbitration. If Alice and Bob simultaneously transmit a certain bit, there are in total four different cases that may occur. These are put together in Table 1. If one of the two nodes transmits a dominant bit (‘0’), the effective bit on the CAN network is also a ‘0’ and only if both nodes transmit a recessive bit (‘1’), we also have the recessive state after the superposition on the CAN network. Therefore, the CAN network may be considered a logical AND function of the individually transmitted bits.

Table 1: Possible combinations of dominant and recessive bit transmissions

Alice	Bob	Effective Bit on CAN Bus
0	0	0
0	1	0
1	0	0
1	1	1

The actual procedure for agreeing on a shared secret between Alice and Bob is a multistep approach as follows:

- Alice and Bob independently generate random bit strings R_{Alice} and R_{Bob} of a predefined length N.
  Example (for N = 10):
  R_{Alice} = 0 1 1 0 1 0 0 1 0 1
  R_{Bob} = 1 0 1 1 0 1 0 1 1 0
- Alice and Bob extend these random bit sequences in such a way that after each bit the corresponding inverse bit is inserted, leading to the modified bit sequences S_{Alice} and S_{Bob} of length *2N*.
  Example:
  S_{Alice} = 01 10 10 01 10 01 01 10 01 10
  S_{Bob} = 10 01 10 10 01 10 01 10 10 01
- Alice and Bob simultaneously transmit the bit sequences S_{Alice} and S_{Bob}, leading to the superimposed bit sequence S_{eff} on the CAN network, which is given as S_{eff} = S_{Alice} AND S_{Bob}.
  Example:
  S_{eff} = 00 00 10 00 00 00 01 10 00 00
  Eve may easily determine this bit sequence as well by means of simple passive eavesdropping on the channel. Therefore, this sequence does not provide any direct advantage yet.
- Alice and Bob determine all tuples in S_{eff} which include a ‘1’.
  Example: In S_{eff} given above, there is a ‘1’ in tuples number 3, 7, and 8 (assuming that we start counting the tuples with 1).
- Alice and Bob delete the bits in their original random bit sequences R_{Alice} and R_{Bob} corresponding to the tuples which included a ‘1’ as determined in step 4. The results are two shortened bit sequences, denoted as K_{Alice} and K_{Bob}.
  Example: Based on the outcome of step 4, Alice and Bob have to delete the bits at positions 3, 7, and 8 in their original bit sequences R_{Alice} and R_{Bob}. We get:
  K_{Alice} = 0 1 ~~1~~ 0 1 0 ~~0~~ ~~1~~ 0 1 = 0 1 0 1 0 0 1
  K_{Bob} = 1 0 ~~1~~ 1 0 1 ~~0~~ ~~1~~ 1 0 = 1 0 1 0 1 1 0

This is done because whenever the effective bit on the CAN network is a ‘1’, it is clear that both Alice and Bob must have transmitted a ‘1’. Likewise, since the two bits in a tuple are always inverse to each other, it is also clear that in that case both nodes must have transmitted a ‘0’ for the other bit. However, exactly the same conclusion can be drawn by Eve and therefore the tuples including a ‘1’ do not provide any usable information for us as no secrecy is contained. For that reason, these bits are simply removed from S_{eff}.

- The resulting bit sequence of Alice is now exactly the inverse of the corresponding bit sequence of Bob, which is the established shared secret.

What remains after step 5 are the bits that are different in the initial bit strings R_{Alice} and R_{Bob}. When simultaneously transmitting S_{Alice} and S_{Bob}, we always get ‘00’ for the tuples corresponding to these bits. Hence, by eavesdropping on S_{eff}, Eve only knows that Alice and Bob have inverse bits in their original random bit sequences R_{Alice} and R_{Bob} at that position, but she is not able to tell which one has the zero and which one has the one. Alice and Bob, in contrast, know which bit they have transmitted themselves, they can also conclude that the respective other peer has transmitted the inverse bit by evaluating S_{eff} and therefore they have a clear advantage compared to Eve.

With the proposed scheme it is possible to establish a shared secret between Alice and Bob by means of a simple public discussion, i.e., by simply transmitting and receiving CAN frames and interpreting the superimposed frames on the bus. Consequently, the complexity is extremely low, especially compared to existing key establishment schemes and can be done in an automated manner.

In a practical realization, the simultaneous transmission of the bit strings S_{Alice} and S_{Bob} would be done in the payload part of a CAN frame. It is possible to implement the proposed scheme in such a way that other nodes (apart from Alice and Bob) see valid CAN frames and do not trigger the transmission of any error frame. The number of payload bits in a CAN frame is limited to 64 bits in case of Classical CAN and 512 bits in case of CAN FD and the length of the shared secret is not constant, but depends on how many values are equal in R_{Alice} and R_{Bob}. Therefore, the length of the shared secret that can be obtained from one simultaneous message exchange may vary between 0 and *N*, with an expected value of *N/2*. Since the initial random sequences are extended by a factor of two by inserting the inverse bit after each bit and since all these *2N* bits have to be transmitted over the CAN network, the overall efficiency ρ, which relates the length of the usable shared secret after one round of the proposed approach to the number of required bits to establish this shared secret, is given by

$$0 \leq \rho \leq ½,$$

with an expected value of E[ρ] = ¼. This means that on average four payload bits have to be simultaneously transmitted by Alice and Bob in order to establish one secret bit. To achieve state-of-the-art security, symmetric keys of 128 bit or even 256 bit are required, which is why a single run of the proposed approach is not enough to generate a sufficient number of secret bits. Therefore, suitable protocol mechanisms are required, which enable the generation ▷

of keys of arbitrary lengths. This may be done by repeatedly performing the proposed procedure and combining the secret bits generated during each run.

Our approach cannot only be used to generate keys of arbitrary length, but to periodically refresh existing keys. By doing so, certain attacks become more difficult and the potential damage if a particular key is revealed can be limited. Therefore, periodic key refreshment is a highly recommended security practice ([10], [11]). For refreshing a key, a limited number of secret bits is sufficient as they may be combined with the old key. The procedure can be inserted into regular CAN communication in order to generate new secret bits and to refresh the used keys.

Finally, CAN is a multicast-based communication protocol and messages transmitted by one node usually have to be received by multiple nodes. This implies that in many cases not only the communication between two nodes has to be secured, but rather the communication between groups of several nodes. Hence, cryptographic schemes for message authentication, encryption, etc. may only be applied if all devices belonging to a certain group are in possession of the same cryptographic key. The procedure proposed in this paper, however, cannot be extended to a multi-node setup in a straightforward manner. Nevertheless, solutions for establishing group keys are available. In the simplest case, all nodes of a certain communication group could establish a pairwise key with one particular node of that group and then this node may generate a group key and signal it to all nodes of the group, encrypting it with the previously established pair-wise keys.

Implementation aspects

As already mentioned, the bit strings S_{Alice} and S_{Bob} should be transmitted in the payload field of a CAN frame. Without any additional measures, this may lead to problems and/or compatibility issues. In a direct implementation, the superimposed CAN frame may violate the bit stuffing rule since even if the individual bit strings S_{Alice} and S_{Bob} adhere to this rule, it cannot be assured that this is also the case for the effective bit string S_{eff}. Other nodes may generate an error frame if they observe such a violation on the bus and clock resynchronization may become more difficult.

A solution to fix this problem is to insert a fixed bit change ('01' or '10') in both S_{Alice} and S_{Bob} after each sequence of at most four bits. This way, Alice and Bob would always transmit the same two bits at this position and since the two bits include a bit change, the bit stuffing rule is never violated in the error-free case. This would come at the cost of a higher overhead. Alternatively, Alice and Bob could determine on-the-fly when it is necessary to insert a stuff bit. Both nodes have to read back the effective bit sequence S_{eff} anyway and could thus check if there have been five identical bits and dynamically insert the inverse bit afterwards. Compared to the first solution, the additional overhead would be lower, but the complexity and processing requirements are higher.

A similar problem occurs with the cyclic redundancy check (CRC) field of a CAN frame in case of a direct implementation of the proposed approach. Since S_{eff} depends on both S_{Alice} and S_{Bob}, the valid value for the CRC field of ▷

the effective message on the bus is generally not equal to the superimposed CRC fields of the messages transmitted by Alice and Bob in case that they calculate the CRC field in such a way that the transmitted frames are valid. In order to solve this issue and assure full backwards compatibility to standard CAN, the correct CRC value that matches the effective CAN frame on the bus could also be calculated by both nodes on the fly and then be appended to that frame after the payload field. While making sure that the effective frame on the bus is a valid CAN frame (based on existing specifications), it has the nice side effect that this procedure automatically helps to make sure that both Alice and Bob have received the same effective bit string S_{eff} (which is essential for deriving the same shared secret). If the two nodes are receiving differing bits, they would append different CRC fields, which would be detected by one of the nodes if a recessive bit was overwritten by a dominant one.

With these approaches to dealing with bit stuffing violations and the CRC field, it is possible to achieve full backward compatibility in the sense that all frames on the CAN network are in line with the existing specifications – at least in the error-free case. Smooth migration paths become possible, where not all nodes connected to a CAN network necessarily have to support the proposed approach and existing hardware and software can be reused. We envision flexible implementation options, where existing CAN controllers do not have to be modified as long as they are supplemented by an additional hardware or software module.

Security considerations

If Eve is only passively eavesdropping on the CAN network, she will not be able to readily determine the established shared secret bit sequences K_{Alice} or K_{Bob}: she only knows that the remaining bits were different for both nodes, but cannot tell who has transmitted the zero and who the one. If, in contrast, Eve is trying to perform an active attack, for example by sending additional own bits during the exchange of S_{Alice} and S_{Bob}, two different possibilities have to be considered:

- Eve transmits a recessive bit,
- Eve transmits a dominant bit.

Transmitting a recessive bit is no different from not transmitting at all since a recessive bit does not change the effective state on the CAN network. Therefore, we only have to analyze what Eve might does by superimposing another dominant bit to the bits exchanged between Alice and Bob. To this end, it is important to remember that with our procedure only those bits remain in the final shared secret for which the effective bit on the CAN network was '0' for both the transmission of the original and the inverse bit. Moreover, if Eve transmits a dominant bit, she cannot tell what the status on the CAN network would have been without her transmission. Therefore, we can conclude the following:

Conclusion 1: An active Eve may disturb our procedure in such a way that the generated secret bit strings K_{Alice} and K_{Bob} are actually not equal on both sides. In order to be able to detect such cases, additional mechanisms should be introduced, with which Alice and Bob can verify that they have generated the same bit sequences. This could be done by calculating and exchanging a hash value of these bit sequences.

Conclusion 2: An active Eve is not able to enforce the generation of a particular shared secret between Alice and Bob (which she then would be aware of) and/or to learn the shared secret that is established between both nodes. This is because K_{Alice} and K_{Bob} depend not only on S_{eff} (which may be determined and influenced by Eve), but also on the bits of R_{Alice} and R_{Bob}, which are unequal on both sides, and Eve has no way to determine these.

Conclusion 3: An active Eve may perform a denial-of-service attack by preventing the establishment of a shared secret, for example by continuously sending a frame with the highest priority. However, this threat exists basically for any scheme since an active Eve could block any CAN communication on the bus. In this case, the failsafe mode of all devices should prevent any serious safety-critical impact.

Conclusion and way forward

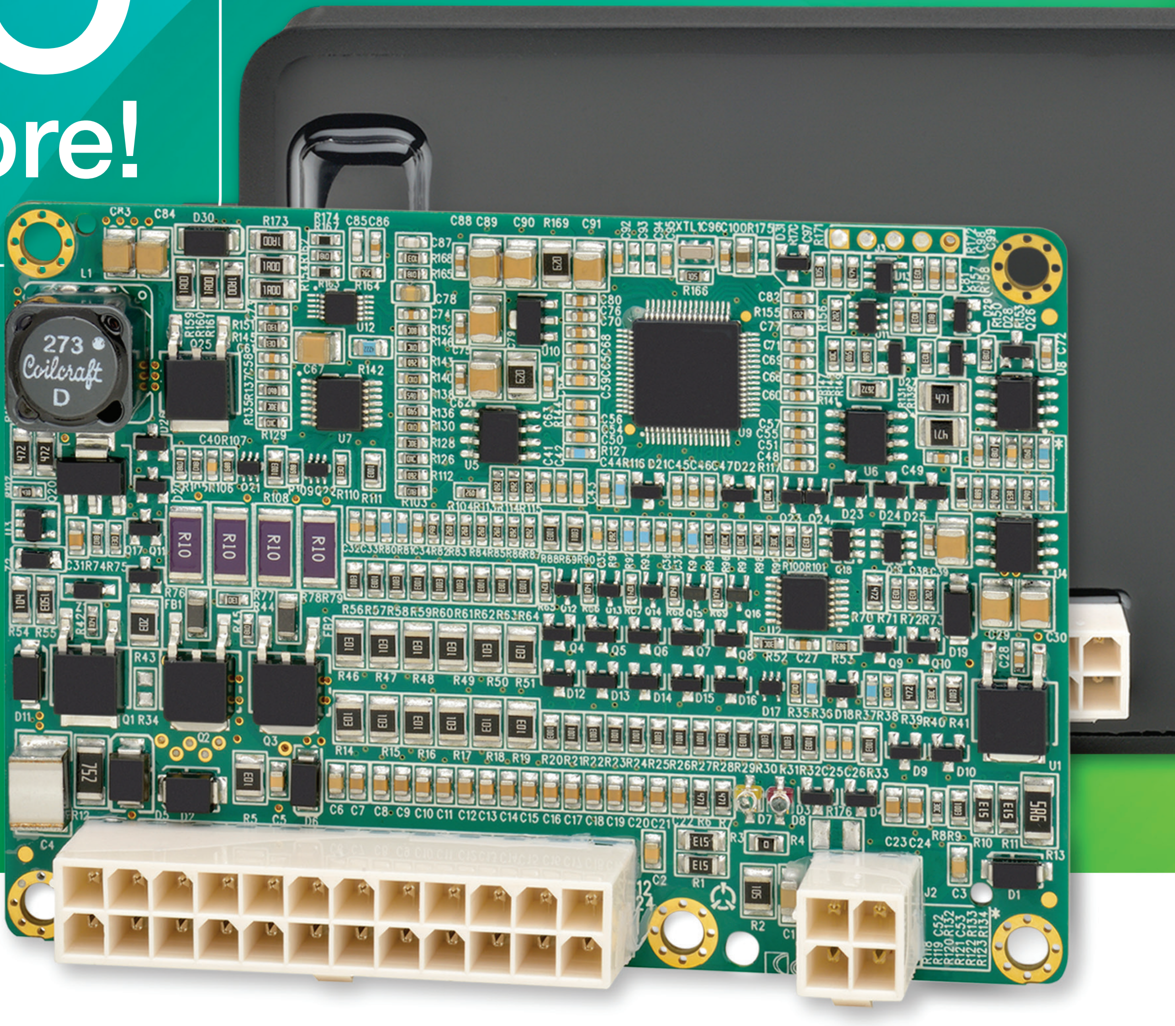
Security will play a crucial role for the success and acceptance of connected systems. A challenge in this regard is how to distribute and manage the cryptographic keys between the involved nodes. We have proposed a novel approach to establishing and/or refreshing symmetric cryptographic keys between two CAN devices in a plug-and-play manner, exploiting special properties of the CAN network. The proposed scheme requires only the simultaneous exchange of random bit sequences along with an appropriate interpretation of the resulting effective bit sequence on the bus. Therefore, it is of very low complexity and may be readily implemented and integrated in practical systems. Even though it cannot address all existing security challenges, it has the potential to become a major building block for secure CAN communication in future. Also, it should be noted that exactly the same concept may also be used in conjunction with other bus systems having similar properties as CAN. Apart from all CAN derivatives, such as TTCAN or CAN FD, this includes the LIN- and I^{2}C-bus.

As a next step, the basic idea has to be embedded in a larger framework, including suitable protocols and mechanisms for synchronized frame transmissions between Alice and Bob, the establishment of group keys, the generation of keys of arbitrary lengths and the like. A first practical proof-of-concept demonstration is planned. In general, no major showstoppers are expected in this respect. ◀

Authors

Andreas Mueller and Timo Lothspeich
Robert Bosch GmbH
www.bosch.com
Andreas.Mueller21@de.bosch.com
Timo.Lothspeich@de.bosch.com

I/O Galore!



THE CURTIS MODEL 1356 CANbus I/O expansion module adds digital, analog and encoder I/O to any CANopen based control system and is ideal for industrial and manufacturing equipment, engine powered and electric vehicles. This low cost, powerful, compact general purpose I/O module, 100 x 70 mm, is available as a PCB or encapsulated. Two models at 12 V-80 V, provide 18 multi- purpose I/O pins with: 13 digital and 5 analog inputs; 1 and 3 Amp outputs configurable as PWM, Constant Current or Constant Voltage; 1 quadrature input; a serial port; 12VDC unregulated and a 5VDC regulated power supplies. The encoder input is suited to reading quadrature encoders from steer or positioning devices with automatic conversion to speed, RPM and distance travelled. The 1356 module is programmed via serial or CANopen for simple integration.

See for yourself. Contact Curtis to expand your I/O.
www.curtisinstruments.com
To see product information and datasheet go to: **www.bit.do/Curtis1356**

1960 · 2015 55 YEARS CURTIS

CURTIS