SECURE HIGH LEVEL COMMUNICATION PROTOCOL FOR CAN BUS

Roman Došek, Peter Janků, Michal Bližňák, Pavel Vařacha

Faculty of applied informatics, Tomas Bata University in Zlin, Nad Stranemi 4511, Zlin 760 05, Czech Republic

Abstract

The Controller Area Network (CAN bus) is a bus based on differential signalling originally developed for automotive industry. The bus was later standardized under ISO 11898 and the standard describes data link layer as well as physical signalling. CAN bus allows precise settings of bus timing and sampling points, which makes it usable for varying ranges and baudrates. It also has a number of properties such as: message acknowledgement, collision avoidance, message filtering and automatic retransmit of faulty messages. These properties make it suitable for many applications. Furthermore, the bus is also well supported on microcontrollers and can even be found on larger SoCs. This makes the CAN bus ideal for microcontroller networks in buildings.
Unfortunately, the CAN protocol itself has no support for node authentication and message encryption so these requirements has to be solved on higher layer. We present a high-level protocol for CAN bus that supports authentication and encryption and therefore allows usage of CAN bus in security dependent systems such as an access management system or in industrial automation.

Keywords: CAN bus; high-level protocol; encryption; access management system

This Publication has to be referred as: Dosek, R[oman]; Janku, P[eter]; Bliznak, M[ichal] & Varacha, P[avel] (2016). Secure High Level Communication Protocol for CAN Bus, Proceedings of the 26th DAAAM International Symposium, pp.1009-1015, B. Katalinic (Ed.), Published by DAAAM International, ISBN 978-3-902734-07-5, ISSN 1726-9679, Vienna, Austria
DOI: 10.2507/26th.daaam.proceedings.142

1. Introduction

Electronic access management systems are getting increasingly popular and are typically included in designs for new buildings. However, many of used solutions lack security against inside attackers and rely on a bad accessibility of communication wires. Because these systems are typically composed of microcontrollers, these are typically connected together through one of the long range communication buses such as TIA/EIA-485, TIA/EIA-422 or CAN bus. Using CAN bus inside this environment can be especially advantageous, because it has a number of features that distinct it from other buses [8]. Each message on a CAN bus is broadcasted through network and there is no addressing and authentication mechanism. These features are however, if necessary, easily solved by using a high-level protocol. Here we investigate requirements for such protocol and a motivation for including these features inside a protocol. Next we lay out a specification of a protocol implementing these requirements and examine its performance in a real environment. Lastly, we discuss comparison of the designed protocol with existing solutions and explore its extensions over other buses.

Field name	Length (*bits*)
Start-of-frame	1
Identifier A	11
Substitute remote request (SRR)	1
Identifier extension bit (IDE)	1
Identifier B	18
Remote transmission request (RTR)	1
Reserved bits (r0, r1)	2
Data length code (DLC)	4
Data field	0-64
CRC	15
CRC delimiter	1
ACK slot	1
ACK delimiter	1
End-of-frame (EOF)	7

Table 1. CAN Frame Structure

1.1. CAN bus frame format

The CAN bus protocol can currently be found in several versions that differ in a frame format. The newest CAN specification - CAN 2.0 specifies two formats - standard format CAN 2.0A with an 11-bit identifier and extended format CAN 2.0B with a 29-bit identifier. Since most microcontrollers available on market offer both of these standards, it makes sense to use the extended variant because it allows to transmit more control bits in single frame, thus reducing arbitration collisions and allowing more flexibility in usage of identifier bits. Further references to CAN bus therefore operate with standard CAN 2.0B.

2. Proposed protocol

2.1. Definition of requirements

The following list describes requirements that were defined for developed protocol along with justification of each requirement.

- Absolute device identification - Each node connected to the network should have its unique ID - equivalent of MAC address in Ethernet. This ID is used to configure relations between devices and to resolve and find dynamic addresses.
- Automatic dynamic addressing - To make absolute address unique, it has to be composed of large number of bits. This makes this address impractical for use during normal network operation because of limited number of bytes that can be transmitted inside single CAN frame. Consequently, each node inside the network has to be addressable by a shorter dynamic address. Since the network can be composed of large number of microcontrollers (64 for example), manual configuration of each node can be difficult and error prone.
- Point-to-point encryption - The network may be composed of devices with different amount of security, so the actual content of message should be visible only to node that possess the valid key. Furthermore, compromise of single key shall not endanger entire network.

- Broadcast - Broadcast is an effective way to implement many of required tasks, such as dynamic address resolving and network scanning.
- Ability to send and identify unencrypted messages - For some of the service messages (dynamic address resolving, network scanning) it makes no sense to enforce encryption. Encrypted and plain messages should therefore be clearly distinguishable.
- Verification of successful decryption - The CAN frame contains CRC-15-CAN checksum which allows the receiving node to quickly check if the data were received correctly. However, this checksum cannot be used to ascertain if the key used to encrypt and decrypt data is the same which may be problematic for transmission of binary data over network. Therefore, the protocol has to offer some other facility which would allow to check origin of received data.
- Frame numbering - Single CAN frame can carry as much as 8 bytes of data. For applications that require sending of larger messages, it is necessary to number frames to determine order of frames and how many frames is remaining (to find if the message is ready for further processing). This feature does not necessarily have to be implemented inside a protocol, but can greatly simplify applications based upon protocol and plays well with its other features.

2.2. Protocol implementation

2.2.1. Frame format:

The length of single CAN frame can range from 64 to 128 bits with service bits taking up more than the actual data in most cases. For this reason, some high level protocols harness bits inside the identifier part of frame (e.g. CANopen, ARINC 825, VSCP). Other protocols put control bits in data part, thus further decreasing the amount of data that can be transmitted in single frame (e.g. CANaerospace). The proposed protocol utilizes the former approach and puts service bits inside the CAN identifier. The extended CAN frame identifier is 29 bits long and is composed of two parts: 11 bits identifier and 18 bits identifier. Both parts are used by proposed protocol. As can be seen from figure 1, the combined identifiers are further split into six fields.

- CMD is 7 bits long and specifies a command. There can be up to 128 commands, where the first 10 are defined by protocol and the rest is user defined.
- REM (as in remaining) is 4 bits long and describes how many fragments of single message are remaining.
- SRC is an 8 bits long source address.
- DST is an 8 bits long destination address.
- ENC (as in encrypted) is a 1 bit specifying if the frames data are encrypted.
- RSV is a 1 bit reserved for future use.

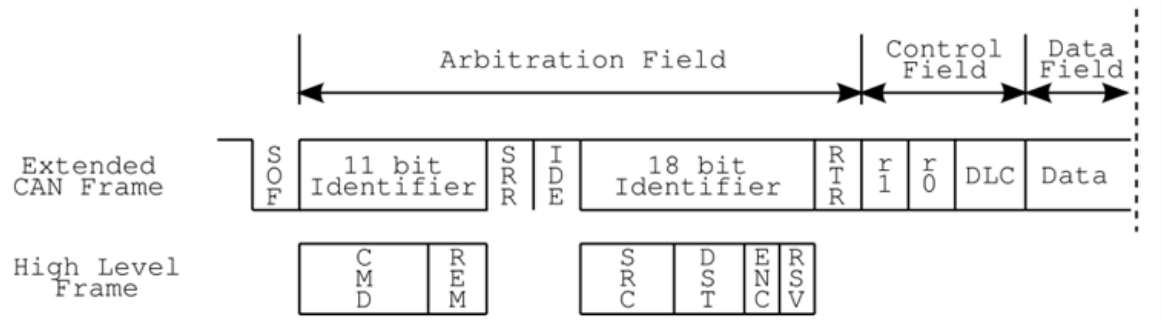

Fig. 1. CAN Frame Structure.

2.2.2. Addressing scheme

Each node has its own unique address that is 8 bytes long. This unique address can either be deduced from serial number of a microcontroller or defined by a manufacturer. The 1 byte long dynamic address is resolved during device startup with the help of other devices on the network. Initially, this dynamic address is calculated through hashing function from the 8-byte unique address. Given the reduction of information from 8 bytes to 1 byte, collisions of this generated address could occur if there was no other mechanism involved. For this reason, after generating dynamic address, the device has to query network through broadcast to find out if the address is not already in use. Because addresses generated this way are not persistent and are re-generated again after each power-up, the network can easily deal with on-demand integration and separation of new devices.

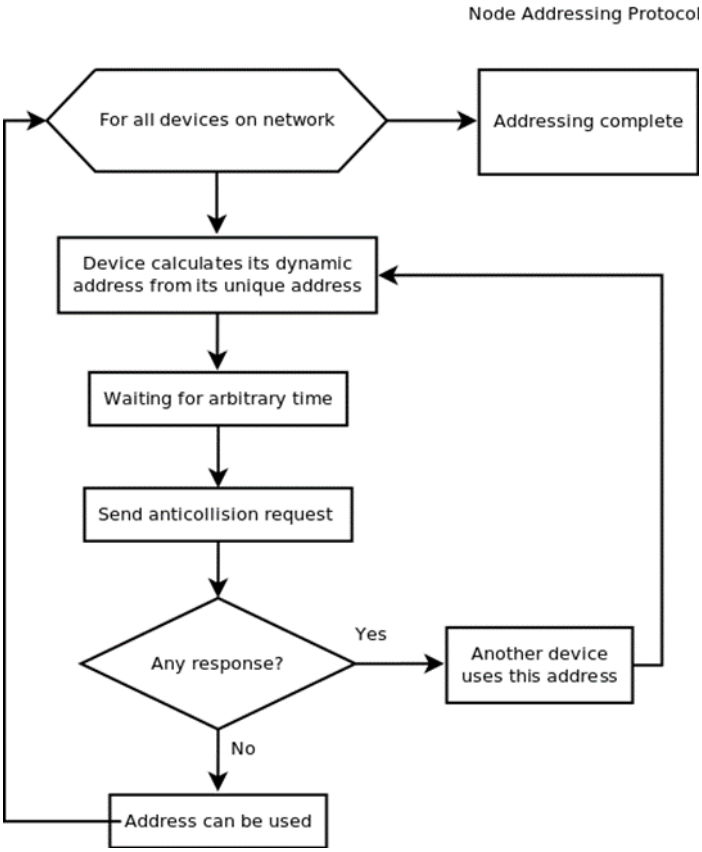

Fig. 2. Dynamic address resolving.

2.2.3. Encrypted session & key exchange

Once the one node wants to start communication with another node, it first has to fulfill two requirements. First it has to know the unique address of the target node and second it has to share a common key with target device. This means that both devices has to be configured to talk to each other and have a randomly generated pre-shared key, that is used for mutual authorization and generation of a session key. First the node that initiates the connection has to discover network address of the target node. This is done by broadcasting frame with command DiscoverNodeRequest and data part that contains unique address of target node. If the network contains such a node, the node is required to reply by broadcasting frame with command DiscoverNodeReply and data part containing its own unique address. The network address is already present in frame header and can be extracted from there.

After both nodes get network addresses of each other, the encrypted session can be initiated. To check authenticity of both nodes and create a session key, variation to the three pass authentication scheme as described in ISO/IEC 9798-2 is used. Following list describes used authentication scheme.

1. A generates a random number RndA, encrypts it using pre-shared key and sends it to B.
2. B decrypts received data, generates a random number RndB, and creates a token RndBRndA' that is comprised from internally rotated RndA and RndB. This token is then encrypted using a pre-shared key and transmitted to A.
3. On receipt of the message containing encrypted token RndBRndA', A verifies B by deciphering message with pre-shared key and then checking that the random number RndA' gained from message is equal to internally rotated RndA sent to B in step (1). If the numbers match, authenticity of B is verified. A now rotates received RndB to RndB', encrypts it and transmits it to B.
4. B deciphers a received message, rotates RndB originally sent in step (2) and compares it to received RndB'. If the numbers are equal, authenticity of A is validated.
5. Because both nodes now possess both RndA and RndB, these numbers can be used to derive session key. The 16-byte session key is gained by taking the first 4 bytes of RndA + first 4 bytes of RndB + last 4 bytes of RndA + last 4 bytes of RndB. This way, if one of the generated random number was full of zeros, the complete key would not be degraded.

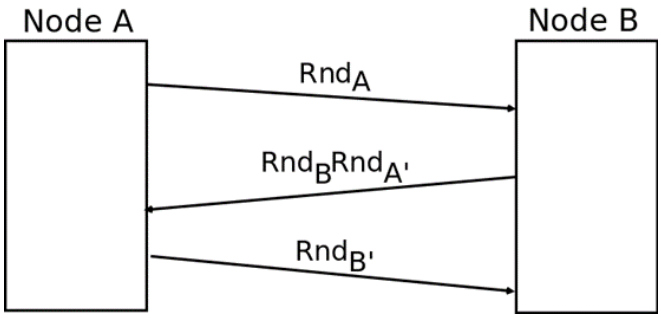

Fig. 3. Tree-way authentication scheme.

2.2.4. Used cipher

When working with microcontrollers, there is often a necessity for compromises due to a memory and processing power constraints. Some ciphers require large lookup tables and are significantly slower without them. And while these issues can be mitigated by using a microcontroller with build-in cryptographic unit, careful selection of cipher is always a necessity. Single CAN frames can contain up to 64 bits of data and this value should ideally be the same as block size of used cipher to avoid unnecessary padding and following transmission of extra frames. Furthermore, when we examine available microcontrollers with build-in cryptographic units, we can find that these units typically offer only DES, TripleDES and AES algorithms. Because AES has block size of 128 bits, each message would have to be padded to multiples of 128 bits, reducing network efficiency for messages composed of odd number of frames. DES cipher is generally considered unsafe due to its key length which is only 56 bits, making brute force attack on cipher feasible. The TripleDES standard defines three distinct keying option with various security [1]. Option number 1 uses 3 independent keys of size 56 bits, however due to Meet-in-the-middle attack [2], [11], [12], the effective security of this options is only 112 bits. Option number 2 uses only 2 keys with length of 56 bits making final key length 112 bits. However due to known attacks based on chosen-plaintext [7] and known-plaintext [10], the final security of this option is designated to be 80 bits. The proposed protocol uses TripleDES cipher with keying option 2 (112-bit key with Encrypt-Decrypt-Encrypt sequence). Multiple blocks of 64 bits are encrypted using Cipher-Block-Chaining (CBC) mode of operation [3]. Note that this option was picked to allow efficient key exchange over CAN bus. Consequently, protocol could be easily modified to use keying option 1 or a different cipher altogether.

2.2.5. Encrypted message format

The protocol uses symmetric block cipher with block size of 64 bits to encrypt content of messages. Messages are therefore always padded with zeros to multiples of 64 bits. For messages longer than 1 block, Cipher-block-chaining(CBC) mode is used. To guarantee that message was deciphered correctly, an 8-byte message authentication code (MAC) is appended as the last frame for each encrypted message. The MAC used inside this protocol is commonly referred to as One-Key CBC-MAC (OMAC) or Cipher-based Message Authentication Code (CMAC) and the core of the algorithm is described in NIST Special Publication 800-38B - Recommendation for Block Cipher Modes of Operation: The CMAC Mode for Authentication [4].

3. Protocol performance and comparison with other networking options

3.1.1. Network throughput for various types of messages

For large networks with large distances between nodes, the CAN bus can only be used with reduced baud rates. It is therefore important to be able to use given bandwidth efficiently to reduce arbitration conflicts and to allow low reaction times. This efficiency is different for encrypted and plaintext messages, because encrypted messages always has to be padded to cipher block size. In addition, each encrypted messages is complemented by one frame CMAC used to verify if the message was decrypted correctly. Each CAN bus frame is composed of framing and actual data. For CAN 2.0B, the total size of framing section is 64 bits. Efficiency for plaintext and encrypted messages can be calculated according to equations 1, 2.

$$E_{Plain} = \frac{Length(data)}{(8 \times (Length(data)/8)) + Length(data)} \tag{1}$$

$$E_{Enc} = \frac{(8 \times Length(data))}{(16 \times (Length(data)/8 + 1))} \tag{2}$$

Figure 4 captures relation between message length and efficiency of transmission. Note that bits used by protocol inside framing part are not included inside calculation as information bits even though it could be argued that they do carry information - about source and target nodes, used encryption and command. However, to make protocol comparable with other high-level protocols that are broadcast oriented, it was decided to omit these bits from the calculation. As can be seen of figure 4, the efficiency is lowest when CAN frame is only partially filled - e.g. 2 of 8 byte of data with the most extreme case being zero transmitted bytes. Efficiency of usage then climbs until it hits 8-byte boundary and another frame has to be transmitted over network. While average effectiveness of plaintext messages is almost 45%, for encrypted messages, this number is significantly lower at around 33%. Loss of efficiency for encrypted messages is caused by a necessity to pad and transmit always 8 bytes of data. Furthermore, each encrypted message is concluded with one full frame containing CMAC. Practical efficiency is of course highly dependent on distribution of message lengths.

3.1.2. Usability of protocol on other buses

The protocol builds upon some of the features of the CAN bus. Nevertheless, it is possible to modify it to work on other buses, such as TIA/EIA-485. On these buses, CAN framing can be emulated by transmitting each byte of frame and actual data separately. However, since these buses are not built to work as multi-master, collisions may occur if all devices are free to transmit.

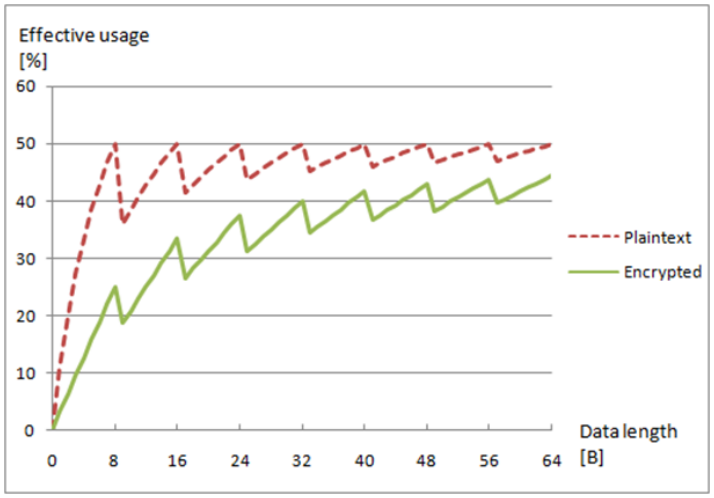

Fig. 4. Relation between length of transmitted data and framing efficiency

3.1.3. Resistance to attacks

While the cipher used in protocol should be theoretically safe for following years, implementation of this protocol may be vulnerable to side-channel attack, especially to differential power-analysis [5]. In this case, keys can be recovered due to current fluctuations during encryption and decryption. These types of attacks can be mitigated by inserting random operations into encryption and decryption process. This protocol is currently vulnerable to replay attacks [9] due to missing key-update mechanism. This problem can be addressed in a number of ways, the simplest one being an extension of the CBC encryption schema over all transmitted and received messages respectively, thus making it impossible for an attacker insert forged frames into communication. In this case, padding of plain-text could be done with random values, thus further diversifying communication and making it harder for an attacker to capture repeating sequence.

3.1.4. Limitations

The proposed protocol conforms to limitation of CAN bus data link layer, specifically maximum length of data in a single frame. This limitation is present due to the original deployment in automotive field, where was no need for longer data. One way bypass it is to use newer version of CAN bus – CAN FD, however this version is not yet widely supported in various microcontrollers.

The second shortcoming comes from usage of block cipher, which forces to align data to single block sizes. One possibility is to use some stream cipher instead and consequently resolve the vulnerability to replay attacks.

4. Conclusion

In this paper, we examined requirements for a secure protocol on a CAN bus along with a justification for these requirements. Next, we presented a protocol that implements these requirements and thus can be used in environments that require secure communication such as access management systems. The proposed protocol was implemented and

tested inside several scenarios comprised of multiple types of devices and no design faults were discovered so far. While there exists other protocol already in use, they are often too specialized for some tasks - lightning control, air flow automation etc. Our protocol allows its user to define custom commands which makes it along with its simplicity an ideal candidate for simpler projects where security matters.
Implementing this protocol allows its user to create dependable and secure network of embedded devices, and to focus on the real application. However, due to need for multiple frames in some scenarios, strict priorities and real-time responsiveness are sacrificed.

For further work, we will focus on proofing the implementation against existing attacks and on solving secure key-update mechanism.

5. Acknowledgements

This paper is supported by the Internal Grant Agency at TBU in Zlin, Project No. IGA/FAI/2015/059 and by the European Regional Development Fund under the project CEBIATech No. CZ.1.05/2.1.00/03.0089 and by the Technology Agency of the Czech Republic as a part of the project called TA03010724 AV and EV LED luminaire with a higher degree of protection.

6. Appendix A. List of reserved commands

Command ID	Command name	Description
1	IDAnticollisionRequest	Transmitted by a node after it generates its dynamic address.
2	IDAnticollisionReply	Broadcasted on a network by a node if it receives AnticollisionRequest with its address.
3	DiscoverNodeRequest	Broadcasted along with an unique address of target node to discover its dynamic address.
4	DiscoverNodeReply	Transmitted by a node that detects DiscoverNodeRequest with its unique address.
5	Accepted	Generic acceptance reply.
6	Rejected	Generic rejection reply.
7	SessionRequest	Initiates first stage of three-way authentication.
8	SessionReply	Second step of three-way authentication.
9	SessionCheck	Last step of three-way authentication.
10	InvalidSession	Used to inform another node that CMAC check failed

Table 2. Reserved commands

7. References

[1] W. C. Barker and E. B. Barker. Sp 800-67 rev. 1. recommendation for the triple data encryption algorithm (tdea) block cipher. 2012.

[2] O. Dunkelman, G. Sekar, and B. Preneel. Improved meet-in-the-middle attacks on reduced-round des. In Progress in Cryptology-INDOCRYPT 2007, pages 86–100. Springer, 2007.

[3] M. Dworkin. Recommendation for block cipher modes of operation. methods and techniques. Technical report, DTIC Document, 2001.

[4] M. Dworkin. Recommendation for block cipher modes of operation: The CMAC mode for authentication. US Department of Commerce, Technology Administration, National Institute of Standards and Technology, 2005.

[5] P. Kocher, J. Jaffe, and B. Jun. Differential power analysis. In Advances in Cryptology?CRYPTO?99, pages 388–397. Springer, 1999.

[6] Jiang, Wei, Zhenlin Guo, Yue Ma, and Nan Sang. "Measurement-based research on cryptographic algorithms for embedded real-time systems." Journal of Systems Architecture 59, no. 10 (2013): 1394-1404.

[7] R. C. Merkle and M. E. Hellman. On the security of multiple encryption. Communications of the ACM, 24(7):465–467, 1981.

[8] O. Pfeiffer, A. Ayre, and C. Keydel. Embedded networking with CAN and CANopen. Copperhill Media, 2008.

[9] P. Syverson. A taxonomy of replay attacks [cryptographic protocols]. In Computer Security Foundations Workshop VII, 1994. CSFW 7. Proceedings, pages 187–191. IEEE, 1994.

[10] P. C. Van Oorschot and M. J. Wiener. A known-plaintext attack on twokey triple encryption. In Advances in Cryptology?EUROCRYPT?90, pages 318–325. Springer, 1991.

[11] P. C. Van Oorschot and M. J. Wiener. Improving implementable meet-in-the-middle attacks by orders of magnitude. In Advances in Cryptology—CRYPTO'96, pages 229–236. Springer, 1996.

[12] B. Zhu and G. Gong. Multidimensional meet-in-the-middle attack and its applications to katan32/48/64. Cryptography and Communications, 6(4):313–333, 2014.