Scalable CAN security for CAN, CANopen and other protocols

Olaf Pfeiffer, Embedded Systems Academy GmbH
Christian Keydel, Embedded Systems Academy GmbH

Commonly used security methods for authentication and encryption/decryption on the Internet cannot easily be applied to CAN/CANopen. The CANcrypt framework described in the book "Implementing scalable CAN security with CANcrypt"[1] adds different levels of security features to CAN. The CANcrypt system is protocol independent and can be used with CANopen or other higher-layer CAN protocols. A manager / configurator is only required for the generation and exchange of keys, but not during regular operation. For key generation, CANcrypt uses a CAN feature that allows two devices to exchange a bit not visible to other CAN devices. This allows generating pairing keys that only the two participants know.
Per default, CANcrypt uses a dynamic 64-bit key to cover the longest possible secure data block, 8 bytes. From this key, a pseudo one-time pad is generated and changes frequently. How often new random bits are introduced to modify the shared key is configurable. 128-bit keys for AES-128 are also supported.
CANcrypt provides a security infrastructure for CAN where developers can still select or customize specific security functions. It can be integrated into existing code at the lower driver level, making it independent from protocol or application layers above.

This paper summarizes the technical methods and features that are used in the CANcrypt security system introduced in the book "Implementing scalable CAN security with CANcrypt" [1].

Limits of CAN specific security

When looking at security for existing CAN based communication then some cases can be excluded from further analysis. These are the cases that either we cannot protect a system from or cases for which there are already a number of solutions available.

If an intruder has access to a CAN system at a level where they can inject any CAN message at any rate, it allows them to run a denial of service style attack by flooding the CAN bus with high priority messages. It then becomes unusable for other participants which has a similar effect as physically cutting the CAN signal lines. In other words, once the door is open to an intruder, a complete shutdown of the system cannot be prevented. However, typically an intruder would not want to break a system but instead extract or manipulate status or control information. Safeguarding the integrity of a system against this type of attack therefore has a higher priority.

For larger blocks of data there are common end-to-end security and encryption standards used on the Internet such as SSL. They can be applied to CAN communications, too, but only in combination with a peer-to-peer transport protocol on top of CAN such as CANopen segmented SDO transfer where larger blocks of data are split into small segments that fit into single CAN messages.

One challenge securing "generic" CAN communications is to develop a security mechanism that can be applied to a single message which includes for example sensor data of just a few bytes, or even single-bit commands that each control an individual switch like unlocking a door, and that also includes those messages that make use of the broadcast feature of CAN by having multiple receivers (one-to-many). Common Internet security protocols are not suitable for these scenarios.

Typical CAN attack vectors and security requirements

An attacker who has gained CAN access to a system was either able to physically install some sniffer device or achieved access to a CAN-connected device remotely. Either way, the attacker should be assumed to be able to receive and transmit CAN messages on the bus.

In many CAN systems, authentication is the only security requirement, addressing questions such as: How can we verify that a received message was really transmitted by the authorized sender and was not injected by some intruder? How can we detect if an intruder disabled an existing CAN device and tried to replace it by mimicking its communication behavior?

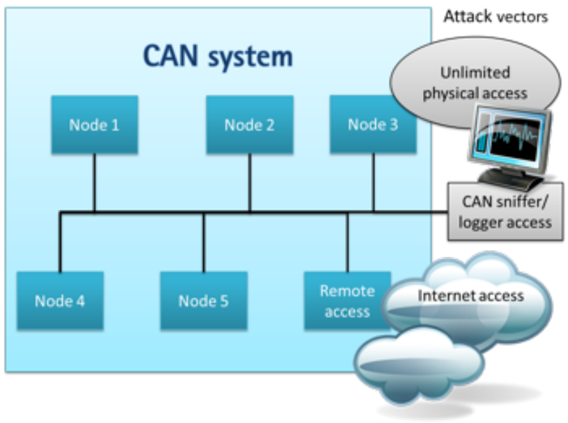

Figure 1: An attacker's access options

Often in CAN systems, encryption is less important. In an industrial or automotive system it is generally more important that a control command or sensor data is authenticated because the data itself is not regarded a "secret". A use case for encryption might be configuration data, which however tends to be bigger than the average CAN message, allowing common Internet encryption methods to be used, as previously pointed out.

Manipulation detection

Already today, activities originating from possible attacks can sometimes be detected as a "side effect". The CANopen [2] application profile CiA 447[3] for add-on car electronics for example has a built-in protection against "spoofing" of nodes. In CiA 447 with its highly dynamic nature, node IDs can be re-assigned upon every system wakeup. In order to avoid accidental duplicate node IDs, every device must monitor the network for CAN message IDs that it transmits itself. If such a message is detected, the device issues an emergency message. So if an intruder injects a message "owned" by another device an appropriate emergency message would immediately show up on the network, invalidating all its communication.

For a successful attack on such a system, an intruder would need to disable the node "owning" (transmitting) the CAN messages in question first, before introducing a device that mimics the behavior of the node.

What to authenticate

Any security feature will add some overhead to the communication. Such added security data includes a secure checksum and housekeeping values as well as messages to maintain the overall security mechanism. This eats up valuable bandwidth, and potentially slows down communication.

Thinking about a minimal authentication feature, all we will need is a secure heartbeat message though, as long as we add it to a system where all nodes monitor their own messages like in the CiA 447 example described above. Each individual message does not need to be protected if it is continuously monitored and manipulations are reported or cause a secure heartbeat timeout.

Minimal authentication with self-monitoring and a secure heartbeat

Devices participating in the secure communication scheme are grouped. The security foundation is a shared symmetric key from which a dynamically changing key is generated.

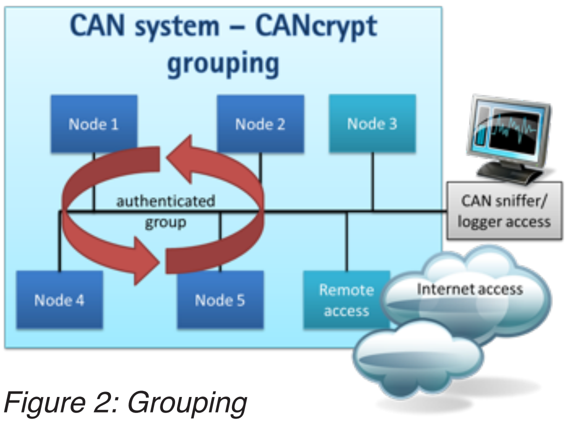

Figure 2: Grouping

Each participating CAN device monitors the network for injected/duplicate messages that use a CAN ID that it itself uses for transmissions. As long as no injection or manipulation is detected, the device keeps producing a secure heartbeat. Otherwise, it produces an emergency or alert message.

On the receiving side, secure messages are only authenticated with the reception of the following secure heartbeat. The injection/ manipulation detection alone is not enough, as we cannot guarantee that the emergency message is successfully transmitted; after all, an intruder could try to specifically block that message with collisions. Only a successfully received secure heartbeat authenticates "all previous messages" from a device.

The disadvantage of such a system is that the secure heartbeat cycle time directly impacts the system control cycle time and therefore needs to be faster than a typical heartbeat period such as the one used in CANopen. Depending on the authentic-cation requirements of a system, a control unit must wait for the secure heartbeat timeout to decide if the previous messages are authenticated. This calls for a secure heartbeat period as short as 100 milliseconds or less.

For completeness please note that due to transmit and receive FIFOs and processing delays, a secure message received just prior to a secure heartbeat might still be an injection. However, that would still be detected as such by the transmitter of the secure heartbeat who would immediately produce an emergency or alert and stop producing the secure heartbeat.

Key management challenges

As with any security system, the management of the used keys can be a tougher challenge than applying the security methods to the communication. Assuming the use of a shared (symmetric) key for the main security features, the question arises when and how this key finds its way into the individual devices and where we keep copies of it. Another challenge is the dynamic update/modification of the shared key. If all participating devices continuously update the key, the specific security algorithms do not need to be very strong. An often updated key creates a one-time pad that even with a simple XOR algorithm produces a very high security level. The question is: How good / random is the one-time pad?

Key hierarchy

With a key hierarchy a device can have multiple authorization levels. The original manufacturer could use the highest priority key – only this key would allow a factory reset or to activate a possible bootloader to re-program the flash memory in the device. The next priority level down could be the "system integrator" level which would allow combining devices from multiple manufacturers in one system. The key for this level would allow restoring a system default configuration. And another priority level could be for the "owner" or technician that needs to be able to replace a single device within the system. This authorization level would support pairing and grouping of the components in a system.

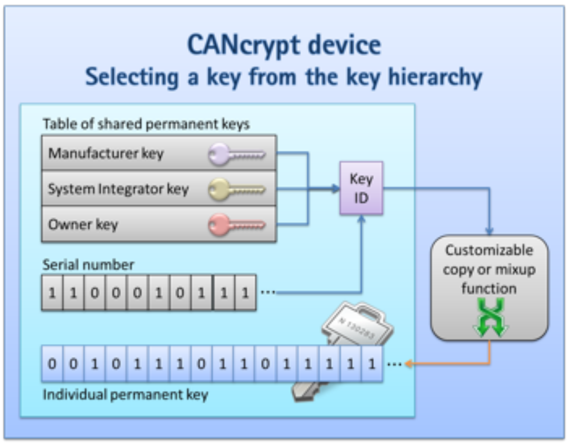

Figure 3: Key hierarchy

Preferably, the lowest priority key level used to pair/group multiple components would not be stored anywhere outside the system. Upon original pairing/grouping, a random shared (symmetric) key would be generated and exchanged between the participating devices and stored locally by each device.

Figure 3 shows that optionally the device's serial number may also be included in the generation cycle for the initial permanent key, which would be used as a base for the dynamic one-time key pad used.

Secure key exchange

The method used to exchange keys described in the following does not use any "direct" CAN communication. Anyone just monitoring the exchanged CAN messages will not be able to determine the values exchanged. The method is loosely based on principals introduced by a paper from Bosch [4] at the last iCC.

By monitoring CAN communications at the message (data link) level, an observer cannot determine the physical device that sent an individual message, because in CAN, any device may transmit any message. As an example, let us allow two nodes (named "configurator" and "device") to transmit messages with the CAN IDs 0010h and 0011h and data length zero. The bits transmit within a "bit select time window" that starts with a trigger message and has a configurable length, for example 25 milliseconds. Each node must randomly send one of the two messages at a random time within the time window.

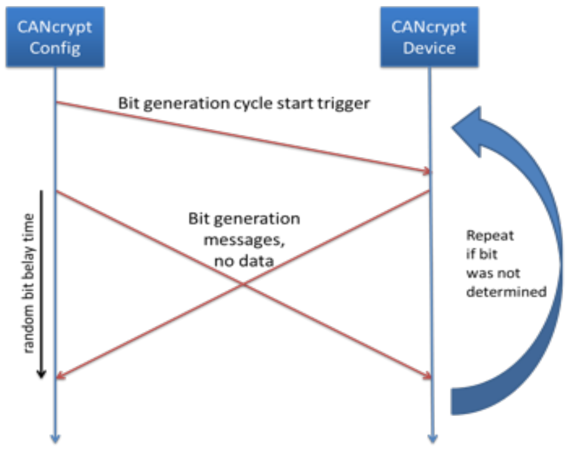
CANcrypt Config
CANcrypt Device
Bit generation cycle start trigger
random bit belay time
Bit generation messages, no data
Repeat if bit was not determined

Figure 4: Bit generation cycle

At the end of the bit select time window, a trace recording of the CAN messages exchanged will show one of the following scenarios:

1. One or two messages of CAN ID 0010h
2. One each of CAN ID 0010h and 0011h
3. One or two messages of CAN ID 0011h

Note that if two identical messages collide, they'll be visible just once on the network. If 0010h and 0011h collide, 0010h is transmitted first followed by 0011h (basic CAN arbitration).

Let us have a closer look at case 2 – one each. If the messages are transmitted randomly within the bit response time window, an observer has no clue as to which device sent which message. However, the devices themselves know it! Now a simple "if" statement can determine the random bit for both participants:

```
IF I am the configurator device
  IF I transmitted 0010h and also saw a 0011h
    common bit is 0
  ELSE IF I transmitted 0011h and also saw 0010h
    common bit is 1
  ELSE
    both used same message, no bit determined
ELSE I am a device
  IF I transmitted 0010h and also saw a 0011h
    common bit is 1
  ELSE IF I transmitted 0011h and also saw 0010h
    common bit is 0
  ELSE
    both used same message, no bit determined
```

Unfortunately, we cannot use cases 1 and 3, so if those happen, both nodes need to recognize it and retry (try again in the next bit select time window).

To prevent an observer from identifying individual device delays, each device should choose two good random values for each cycle. The devices should randomly pick one of the two messages (0010h or 0011h) and randomly select a delay from zero to two-third of the bit select time window.

There are several options to optimize this cycle as well as allowing one device to "enforce" a certain key to the other. Depending on version and timeout, this method can be used to exchange a key of 64 bits within about one second.

Dynamic one-time keypad

All devices participating in the secure communication use a locally stored symmetrical key as a basis. During initialization and detection of their communication partners, the participating devices also exchange random numbers. The combination of all random numbers exchanged during the initial detection is used to generate the initial one-time keypad. This ensures a unique shared initial dynamic one-time keypad with every system start.

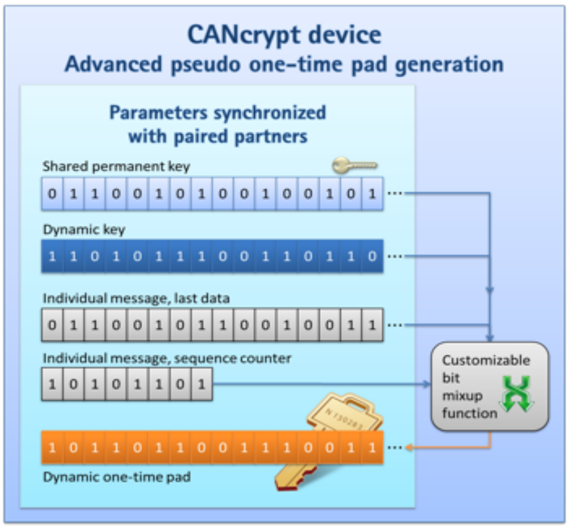

Figure 5: One-time pad generation

Once the devices are paired or grouped, the shared dynamic one-time key pad gets periodically updated, for example using random values from the secure heartbeat and an optional message counter.

Secure heartbeat implementation

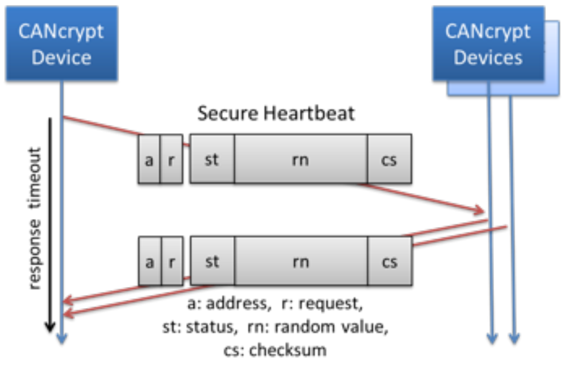

Figure 6: Secure heartbeat contents

All devices participating in the secure communication produce a secure heartbeat. The secure heartbeat is synchronized, in one cycle all participants transmit their own security heartbeat once.

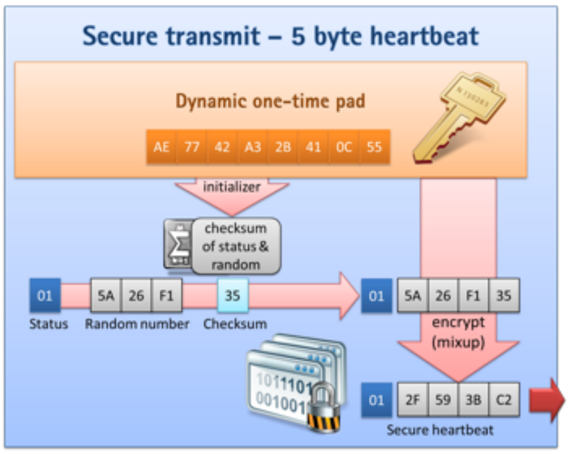

Figure 7: Secure heartbeat transmit

The main component of each secure heartbeat is a three-byte random value with a one-byte checksum. All four bytes are encrypted based on the current shared dynamic key.

All receivers decrypt the four bytes and verify if the checksum matches. If it does, the heartbeat is considered “confirmed”.

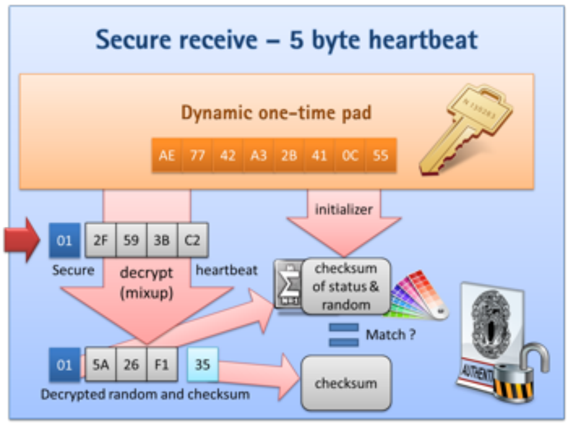

Figure 8: Secure heartbeat receive

All (decrypted) random values of all participating nodes are used to update the shared one-time key pad which is then used for the next cycle. This ensures changes to the dynamic shared key with every use.

In CANopen [2], the secure heartbeat fits into the manufacturer specific fields of the emergency message. This allows the implementation of the secure heartbeat as a variation of the no error / emergency reset message already defined in CANopen.

Point-to-point communication

When using pairing instead of grouping, CANcrypt supports more advanced methods involving individual message authentication and encryption. In this case, individual messages are encrypted and authenticated with a preamble message that contains the security overhead information for the following message.

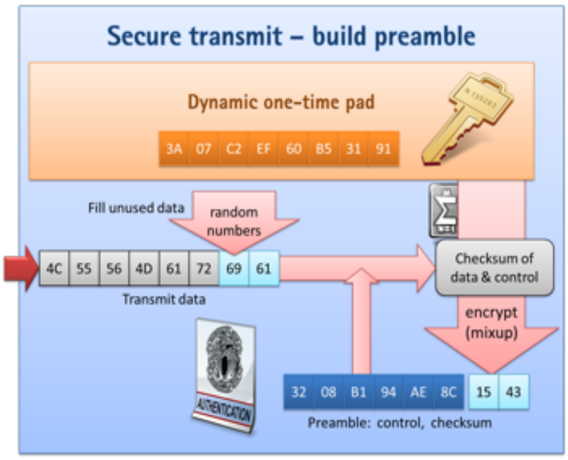

Figure 9: Secure transmit with preamble

Figure 9 shows how a preamble is build. Unused data of the original message is filled with random values. The preamble contains a checksum covering both messages and control/status values. Per default, both the preamble with the checksum and the main message are encrypted. As the total data size is 128 bits, algorithms supporting a 128-bit key size may be used (for example AES-128).

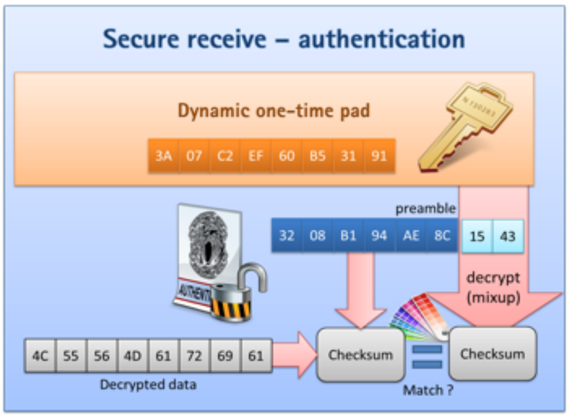

Figure 10: Secure transmit with preambl

Figure 10 illustrates the receiving side. Both the preamble and data message are decrypted and then analyzed. Only if the checksum matches is the message considered “authorized” and passed on to the receive FIFO.

Implementation notes

Looking at existing systems already in use, one of the challenging questions is how security can be added with minimal changes to the software. Often, security is attempted to be added to a higher communication layer. However, the higher up this is added, the more changes to existing software very close to the application level are needed.

The implementation for CANcrypt on the other hand can happen entirely at the driver level with the only requirement being that FIFO buffers are used. To all software layers above the driver, CANcrypt is largely transparent and no software changes are required on the application level. The application will simply not receive data that is not secure as the driver will only pass on messages that are authenticated.

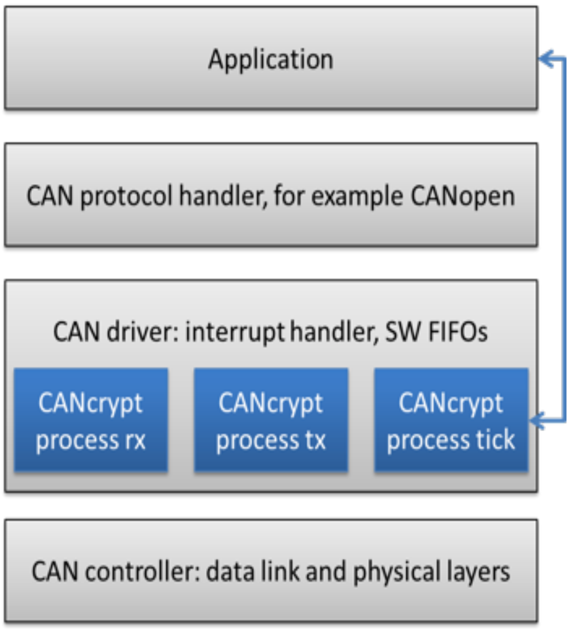
Application
CAN protocol handler, for example CANopen
CAN driver: interrupt handler, SW FIFOs
CANcrypt process rx
CANcrypt process tx
CANcrypt process tick
CAN controller: data link and physical layers

Figure 11: CANcrypt processes

Figure 12 shows in more detail how the CANcrypt processes are incorporated into a typical embedded system using one receive and one transmit FIFO.

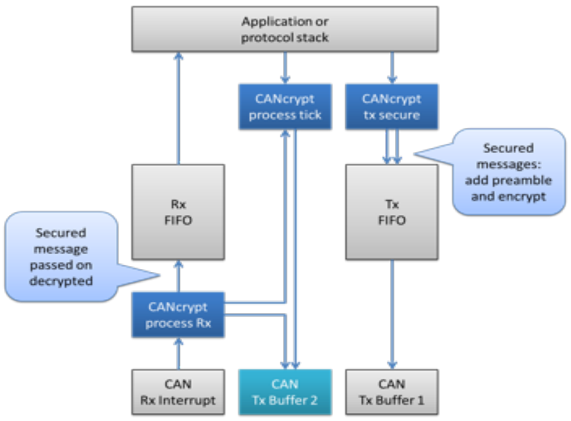

Figure 12: CANcrypt process integration

The main entry points for CANcrypt are on the receive side, before a message is added to the receive FIFO (so CANcrypt can only insert the message if it is considered "secure", i.e. authenticated) and on the transmit side, also before it gets inserted into the transmit FIFO (to possibly add security overhead like a preamble),

CAN-FD or higher-layer protocols

The functions introduced by this paper apply to both CAN and CAN-FD. Where the CAN implementation is based on key lengths of 64 bits and 128 bits, a CAN-FD adaptation may also use keys with a length of 256 bits.

The implementation is independent of the protocol as it can be done on the driver level. Unsecure (not authenticated) messages can simply be hidden from the application.

Summary

This paper highlights the key technology and features used by the CANcrypt system. CANcrypt is a security framework that still allows developers using it to select the individual security methods or algorithms used. These range from a simple grouping with authentication-only to a more secure pairing of two nodes with both authentication and encryption.

CANcrypt security is based on symmetric keys of 64 or 128 bits and data sizes of up to 128 bits (dual message). All security algorithms suitable for a 128-bit data/key length can be used, including AES-128 or comparable algorithms for the security.

References

[1] Implementing scalable CAN security with CANcrypt, book by Olaf Pfeiffer
[2] CiA 301, CANopen application layer and communication profile
[3] CiA 447, CANopen application profile for add-on car electronics
[4] Plug-and-secure communication for CAN, paper at 15th iCC by Andreas Müller, Bosch

Olaf Pfeiffer
Embedded Systems Academy GmbH
Bahnhofstr. 17
DE-30890 Barsinghausen
Tel.: +49-5105-582-7897
Fax: +49-5105-584-0735
opfeiffer@esacademy.de
www.esacademy.com

Christian Keydel
Embedded Systems Academy GmbH
Bahnhofstr. 17
DE-30890 Barsinghausen
Tel.: +49-5105-582-7897
Fax: +49-5105-584-0735
ckeydel@esacademy.de
www.esacademy.com