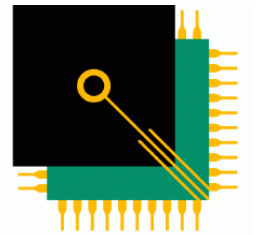
# Implementing Scalable CAN Security with CANcrypt

## Authentication and encryption for the Controller Area Network and CANopen

*by Olaf Pfeiffer*





A technology guide from

EMBEDDED SYSTEMS ACADEMY

Jointly published by

Embedded Systems Academy, Inc.
1250 Oakmead Parkway, Suite 210
Sunnyvale, CA 94085, USA

Embedded Systems Academy GmbH
Bahnhofstraße 17
30890 Barsinghausen, Germany





**Limitation of Liability**

Neither Embedded Systems Academy (ESA) nor its authorized dealer(s) shall be liable for any defect, indirect, incidental, special, or consequential damages, whether in an action in contract or tort (including negligence and strict liability), such as, but not limited to, loss of anticipated profits or benefits resulting from the use of the information or software provided in this book or any breach of any warranty, even if ESA or its authorized dealer(s) has been advised of the possibilities of such damages.

The information presented in this book is believed to be accurate. Responsibility for errors, omission of information, or consequences resulting from the use of this information cannot be assumed by ESA. ESA retains all rights to make changes to this book or software associated with it at any time without notice.

# Available Editions

On its first release in March 2017, this book was made available in two editions:

### *Paperback, limited software version*

**ISBN 978-0-9987454-0-4**

Black and white paperback edition. The demo software matching the examples in the book is available for download at www.esacademy.com/cancrypt. The software license of this software is limited to educational and evaluation purposes.

### *Hardcover, commercial software version*

**ISBN 978-0-9987454-1-1**

Full color hardcover edition including a different, commercial CANcrypt software version delivered using a download code. The software license covers prototyping and an initial pilot production run of up to 500 devices.



**Excerpt Edition**

# About this book

When the Controller Area Network (CAN) was designed, security was not a requirement. The primary usage of CAN was considered closed; possible intruders or attackers would simply not get physical or remote access to the network. However, today it is more and more common that devices connected to a CAN system also have connections to other networks, including the Internet. Recent car hacks have shown that attackers may get access to CAN systems. Without strong security features, an attacker automatically gains full access to everything connected, allowing active control commands to be recorded and replayed.

In this book we examine which options developers of CAN based systems realistically can use to provide adequate security features.

What can we do…

- without introducing heavy-weight security protocols?
- to detect possibly injected messages?
- without any hardware change?
- with minimal software change and integration effort?

We introduce the open CANcrypt protocol and software interface, which provides a scalable and customizable CAN security system. Depending on the application requirements and resources available in the individual devices, various protection levels can be realized.

# Acknowledgements

Writing a technical book is seldom a single-person achievement. I hereby express my gratitude to everyone who contributed to this book. Some did this knowingly, others unknowingly.

With his publications and talks, Klaus Schmeh (Schmeh) continuously keeps fueling my interest in everything related to cryptography and has done so for many years. His works and examples are a reminder that cryptography comes in many varieties.

The works of Ian Foster (Fast and Vulnerable: A Story of Telematic Failures, 2015) (Exploring Controller Area Networks, 2015), Charlie Miller (Miller, 2015), Chris Valasek (Miller, 2014), Craig Smith (Smith, 2016) and others have revealed various

security vulnerabilities in systems using CAN. Their efforts have drastically increased the overall awareness for CAN-related security issues.

The NSA published the SPECK (Ray Baulieu, 2013) lightweight cipher in 2013. At first I was reluctant to use a NSA published cipher. However, until 2017 none of the many publications about it found any major caveat. A variation of SPECK is used for the default implementation of CANcrypt. However, just to be sure, its use in CANcrypt is on top of dynamically changing keys. An attacker who is able to determine a single dynamic key only has a few seconds (or milliseconds) to exploit it until the next key is used…

The CiA (CAN in Automation user's group) maintains most CAN and CANopen related standards. Participating in many standardization meetings has taught me that occasionally one just has to "do it," otherwise it might take years until first results are available.

My partners and colleagues Christian Keydel, Andrew Ayre and Ralf Kindermann offered continuous support and help with many of the technical aspects of the CANcrypt protocol as well as implementation-specific issues.

The lector for this book project has been Jan Axelson (www.janaxelson.com). Jan is a known technology writer and was a perfect match for this project from the start.

And last, book projects commonly require unusual working hours. And those require support and patience from the loved ones surrounding you. Leah, Magnus, Maria, thank you!

Olaf Pfeiffer
February 2017

# Bibliography

*Achieving Confidentiality Security Service for CAN.* **Chavez, Rosete, Henriquez. 2005.** s.l. : Electronics, Communications and Computers, CONIELECOMP 2005, 2005.

*CANAuth - A Simple, Backward Compatible Broadcast Authentication Protocol for CAN.* **Anthony Van Herrewege, Dave Singelee, Ingrid Verbauwhede. 2011.** s.l. : ECRYPT Workshop on Lightweight Cryptography, 2011.

*Checksum and CRC Data Integrity Techniques for Aviation.* **Koopman, Philip. 2012.** s.l. : Electrical & Computer Engineering, 2012.

**CiA 301. V4.2 2007.** *CANopen Application layer and communication profile.* Nürnberg : CAN in Automation (CiA) e.V., V4.2 2007.

**CiA 305. V2.2.17 2013.** *CANopen Layer setting services (LSS) and protocols.* Nürnberg : CAN in Automation (CiA) e.V., V2.2.17 2013.

**CiA 416. V2.0 2007.** *Application Profile for Building door control.* Nürnberg : CAN in Automation (CiA) e.V., V2.0 2007.

**CiA 447-1. V2.0 2012.** *CANopen Application profile for special-purpose car add-on devices.* Nürnberg : CAN in Automation (CiA) e.V., V2.0 2012. Vol. 1: General Definitions.

*Exploring Controller Area Networks.* **Ian Foster, Karl Koscher. 2015.** 6, s.l. : login www.usenix.org, 2015, Vol. 40.

*Fast and Vulnerable: A Story of Telematic Failures.* **I. Foster, A. Prudhomme, K. Koscher, S. Savage. 2015.** Washington, DC : Proceedings of the 9th USENIX Workshop on Offensive Technologies , 2015.

**Koopman, Philip.** Best CRC Polynomials. [Online] Carnegie Mellon Univeristy. https://users.ece.cmu.edu/~koopman/crc/.

**Miller. 2015.** *Remote Exploitation of an Unaltered Passenger Vehicle.* Las Vegas, NV : Black Hat USA, 2015.

**Miller, Valasek. 2014.** *A Survey of Remote Automotive Attack Surfaces.* Las Vegas, NY : Black Hat USA, 2014.

**Pfeiffer, Ayre, Keydel. 2003.** *Embedded Networking with CAN and CANopen.* San Jose : s.n., 2003.

*Plug-and-secure communication for CAN.* **Andreas Mueller, Timo Lothspeich, Robert Bosch GmbH. 2015.** Vienna : international CAN Conference, 2015.

**Ray Baulieu, Douglas Shors, Jason Smith. 2013.** *The SIMON and SPECK Families of Lightweight Block Ciphers.* s.l. : NSA, 2013. Cryptology ePrint Archive 2013/404.

**Schmeh, Klaus.** Klausis Crypto Kolumne. [Online] http://scienceblogs.de/klausis-krypto-kolumne/.

**Smith, Craig. 2016.** *The Car Hacker's Handbook: A Guide for the Penetration Tester.* s.l. : No Starch Press, 2016.

**Voss, Wilfred. 2008.** *A Comprehensible Guide to J1939.* s.l. : Copperhill Media, 2008.

# Excerpt Contents

# 1 Introduction

The CAN (Controller Area Network) is a 30-year-old communication technology that worked well for much of its history without any security features. So what changed that now some 30 years later we need to review the network's security features?

The original use case for CAN systems was that of a closed network. The network would be deeply embedded in machinery without any connection to other networks or the Internet. In this case, any hacker attack would only be a physical attack – to get access to the CAN subsystem, a hacker would need physical access to the machine.

However, today the CAN subsystem is no longer self-contained. More and more often, bridges and gateways to other networking technologies are added, including connections to devices that have access to the Internet. Some examples of these devices are remote access devices for diagnostics or maintenance and multimedia servers like those in use in some automotive applications.

**When we add devices that implement a gateway to the Internet or offer wireless communication options like Bluetooth and Wi-Fi, we also open a door for possible attacks to the CAN network.**

Once a possible intruder is past the firewalls that limit access, there are typically no further hurdles as all communication is unprotected and in the case of CANopen or J1939, even well documented.

## 1.2 Prerequisites

You should have a reasonably good understanding of how the Controller Area Network works before continuing reading. If you are unsure, we suggest reading (CiA 301, V4.2 2007) or one of the many online "Controller Area Network Primer" documents like
www.computer-solutions.co.uk/download/Peak/CAN-Tutorial.pdf

Embedded Systems programming knowledge is required as soon as you want to start implementing CANcrypt on microcontrollers. In this book you will find pseudo code as well as a description of the C functions that make up the user interface of CANcrypt.

## 1.3 What is at risk?

Some might ask what the specific security risk is. Here the popularity of CAN comes into play. These days almost everything with wheels uses CAN (including electric bikes). Maritime and avionic use is also common. You can find CAN in elevators, medical equipment (including surgery robots), and many industrial control processes. Various "car hacks" have been made public, including odometer manipulation and unlocking doors but also active control of steering and brakes.

CAN is also a popular communication channel for software updates. Often new software can be loaded into components of a system via a CAN channel. This ability is a hacker's dream come true – being able to load software into a device with an embedded system that otherwise would not be accessible.

If these systems can be hacked, and hackers can both read and actively send commands/control, then we have to ask ourselves:

- How many hackable vehicles, ships and planes are out there?
- How many hackable elevators are out there?
- How many hackable medical devices are there?
- Is it possible to manipulate any of the above in a way that someone gets harmed and it is only recognized as "technical malfunction"?

Today some of these questions might still sound like the topic for a Hollywood blockbuster. However, it makes one wonder how Edward Snowden would evaluate the likeliness of such scenarios. Maybe it has already been done and just not published…

## 1.4 Usage with I²C, RS-485 and others

Many of the methods introduced in this book can also be applied to other lightweight embedded networking technologies. However, one of the central elements of CANcrypt is the bit-generation cycle. Here two nodes can generate or exchange a bit without other nodes being able to detect which bit was generated or exchanged. For this method to work, a shared transmission medium is required (not dedicated transmit and receive lines). The shared medium could be a single shared wire or pair of wires, as in RS-485, or the I²C data line in multi-master mode.

If we see enough requests, we will adapt CANcrypt to other suitable communication technologies in the future.

## 1.5 Definitions and Terminology

Cancrypt uses data types as defined in CANopen. Here are some of the common terms used:

### *address, device*

Each device has a unique address. This is comparable to the node ID in some network technologies. CANcrypt supports up to 14 devices using the addresses 1 to 14.

### *bit-generation cycle (or bit claiming)*

Two nodes can exchange a bit without others on the network being able to detect the bit value. This process, which requires multiple messages, is the bit-generation cycle.

### *CAN message ID*

Each message has a unique identifier called the CAN message ID, typically 11 bits, sometimes 29 bits (referred to as an extended identifier).

### *configurator, CANcrypt configurator*

The configurator actively configures individual devices for functions such as key generation, assignment, storage, and erase. The configurator is not required during regular operation and has the CANcrypt address 15.

### *device, CANcrypt device*

A system may have up to 14 devices capable of processing CANcrypt messages.

### *error counter*

Each device or configurator maintains an internal error counter. The count is incremented with every suspicious system behavior, including errors. If the error counter reaches a specified value, secure communication is halted.

### *grouped, grouping*

Every active device is in a group where all devices share a common dynamic key and communicate with each other securely. Messages are authenticated and optionally encrypted and decrypted based on the group key.

### *index, to data object*

All parameters are addressable using an index and sub-index value. This addressing method is adapted from CANopen and other networking technologies.

### *INTEGERxx, data type*

INTEGERxx is the notation for a signed integer value where “xx” indicates the number of bits used by the data type. CANcrypt uses “little endian” notation.

### *key, dynamic*

A dynamic key is the base for all cryptographic functions. This key gets updated frequently and synchronously by all active devices.

### *key, hierarchy*

Implementing of a key hierarchy is recommended to enable a device to have multiple keys with different levels of authority. For example, a manufacturer key might give access to a bootloader, while a system builder key might give access to system configurations.

### *key, management*

As soon as keys get permanently stored in devices, key management is required to determine when each key is generated and by whom. CANcrypt supports multiple models of key management.

### *key, permanent*

At least one key must be permanently stored in each device. If a device supports storing multiple keys in a hierarchy, the term permanent key refers to the stored key that is currently in use.

### *message, secure*

CANcrypt transmits each secure message paired with a preamble message that announces the secure message that follows. The secure message contains all of the data in the original (unsecure) message, possibly encrypted.

### *message table*

The message table is shared by all active devices. It lists all CAN messages that require security handling. A device that receives a CAN message listed in this table may pass the message to the application only if the message was received with the matching preamble and signature. For details see section **Error! Reference source not found.Error! Reference source not found.**.

### *one-time pad, pseudo*

For all encryptions and authentications a pseudo one-time pad is used. This one-time pad is available to all grouped or paired devices so that it can be used like a synchronous key.

### *paired, pairing*

Two active devices may get paired to have an individual secure communication channel. Messages are authenticated and optionally encrypted and decrypted based on the pairing key.

### *preamble*

The preamble is a CAN message that announces the secure message that follows. The preamble contains control data as well as a signature that covers the control data and the secure message.

### *STRING4, data type*

STRING4 is the notation for a string segment containing four ASCII characters. The string is zero-terminated. All unused bytes of this string are set to zero.

### *sub-index, to data object*

All CANcrypt parameters are addressable via index and sub-index values. This method to address data is adapted from CANopen and other networking technologies.

### *UNSIGNEDxx, data type*

UNSIGNEDxx is the notation for an unsigned integer value where “xx” indicates the number of bits used by the data type. CANcrypt uses “little endian” notation.

# 2 Selecting cryptographic methods

Many CAN devices are based on microcontrollers with limited memory and processing power. Yet at the highest supported speed of 1 Mbps, the CAN message rate can be as high as 10,000 messages per second. At this rate, adding reasonably safe security software to existing devices may be a challenge.

As in most security systems, there is a tradeoff between how much security we need vs. how much we can afford in terms of resources that we can spare.

## 2.1 Choosing cipher algorithms

Until recently, even the smallest, lightweight ciphers like Blowfish still required minimal block or key sizes of 128 bits and a substantial number of processor cycles to execute. Since the introduction of the Speck lightweight cipher block sizes down to 32 bits are possible and the algorithms are well suited to be handled by limited performance microcontrollers. By itself, all of these security algorithms do not protect from a simple monitor, record, and replay attack.

Note that even the simplest cipher algorithm like a single exclusive OR (XOR) is considered unbreakable (literally safer than anything commonly used today) if the key is as big as the data and only used once. This is referred to as the one-time pad cipher.

So for a 32-bit value transferred, if we use a single one-time 32-bit key combined with a single XOR, we already have an encryption stronger than any other cryptography method in use today.

To a certain extent (depending on how much communication overhead is used and how often), the CANcrypt system introduced in this book allows providing such an individual key. However, “the best protection available” is hardly required for CAN communication. So even if the same or just a slightly different key is used a few times, the protection would still be adequate.

A general rule for many security systems is that the more often you use a key, the more data a possible attacker has available to analyze what is happening. If keys are only temporary and never used again, the attacker has little to work with.

The configurable CANcrypt system uses the following security features:

- Configurable and customizable algorithms for
  - Generation and update of one-time pad
  - One-time pad generation based on Speck or AES-128 (Advanced Encryption Standard)
  - Checksum / hash calculation for authentication
  - Encryption and decryption
- Secure message size is 128 bits (two CAN messages)
  - Supporting 128-bit based algorithms such as AES-128
- All keys are synchronous and shared among two or multiple communication partners
  - Current key used is the dynamic key, which changes after every use (used to generate pseudo one-time pad).
  - Permanent keys (hard coded or stored in non-volatile memory) are used for initialization of the dynamic key
  - Support of a key hierarchy (manufacturer, integrator, owner)

## 2.2 Elementary function: bit generation

The elementary functionality that CANcrypt provides is the generation of a bit that is known to two communication partners but not visible to anyone else. This can be a random bit, or one of the communication partners can enforce a bit. Two devices can use the bit to secretly exchange (or generate) a key. As this operation can occur at any time during operation, keys can become dynamic: new bits are introduced or added to the shared key continuously during the operation.

With this base functionality, we can pair two devices, and if the main shared key is continuously updated, the encryption, decryption, and authentication algorithms may be minimal. If the key changes randomly, an attacker that has no access to the bit generation will barely have any data to work with.

In summary, for CANcrypt the focus is not on the cipher algorithm but on the key. In the default dynamic key mode, a 64-bit key (to cover the longest possible secure data block of eight bytes) is used. The key is modified after every use. The CANcrypt configuration determines how often new random bits are introduced into this key modification.

### 2.2.1 The bit-generation cycle

When monitoring CAN communications on the message level, one cannot determine the device that sent an individual message because any device may transmit any message. As an example, let us allow two devices (named dominant device and recessive 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 ms. Each node must randomly send one of the two messages at a random time within the time window.

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

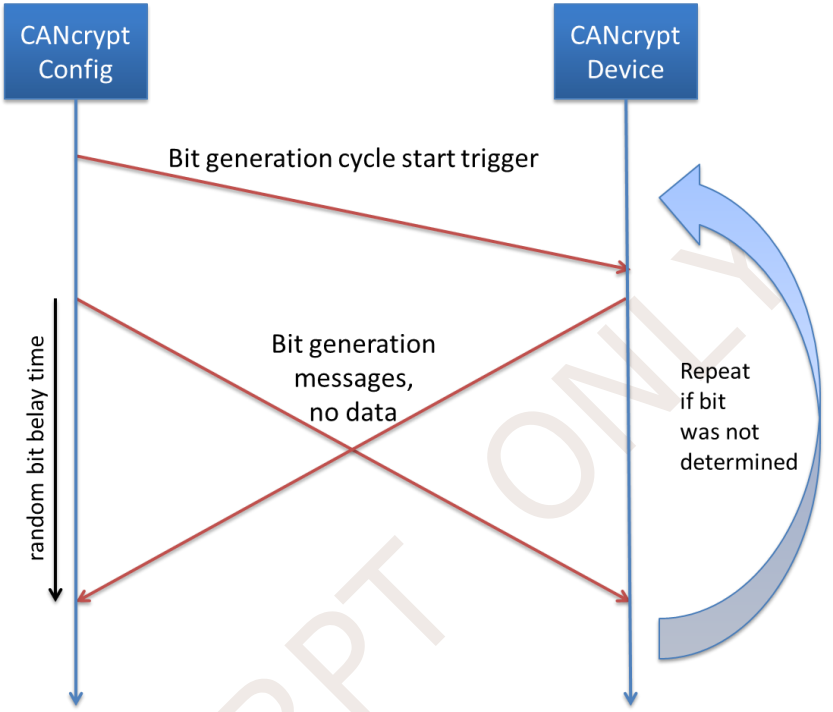
1. One or two messages of CAN ID 0010h
2. One each of CAN ID 0010h and 0011h
3. One or two messages of CAN ID 0011h

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
```




THE BIT-GENERATION CYCLE

Unfortunately we cannot use case 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 0 to 2/3 of the bit select time window.

### *Higher-performance variations*

A variation of this scheme is to not use a random delay but instead ensure that both devices directly transmit their message after the trigger message. Then both messages arbitrate the bus at the same time. In a trace recording, we will always see 0010h followed by 0011h. This scheme requires very fast reactions from the two microcontrollers using the method. From the CAN receive interrupt to the

next transmit trigger, there are only a few bit times (inter frame space, seven bits), so at 1 Mbps this is just a few microseconds.

In order to minimize the chance that both devices select the same bit generation message, a variation of the scheme can use 16 or more different CAN IDs for the bit generation message. Here each device randomly selects one of the 16 messages for the bit generation. Statistically the chance that both devices select the same message is now reduced from 50% to 6%. The average duration of the complete bit-generation cycle thus shrinks drastically. The bit generation algorithm changes slightly to:

```
IF I am the configurator device
  IF I transmitted lower bit generation message
     common bit is 0
  ELSE IF I transmitted higher bit generation message
     common bit is 1
  ELSE
     both used same message, no bit determined
ELSE I am a device
  IF I transmitted lower bit generation message
     common bit is 1
  ELSE IF I transmitted higher bit generation message
     common bit is 0
  ELSE
     both used same message, no bit determined
```

### 2.2.2 Protection level achieved

At the logical (message) level, a bit generated or exchanged is invisible to the other communication partner. If all you can see is the CAN messages exchanged, then by monitoring CAN messages, you will not be able to determine the bit exchanged or generated.

However, an attacker who has full physical access on a signal level (oscilloscope) or on the transceiver level (connection to the microcontroller) can see which node sends which bit-select message. Nevertheless, this access only provides partial information. In CANcrypt, configurable factors, including the stored permanent key, determine how a bit is finally generated or selected.

A note in regard to random bit generation: As usual when it comes to randomness, both communication partners require a reasonably good random generator with an appropriate seed value. (If the initial seed is predictable, so is the randomness.)

## 2.3 Keys: generation, hierarchy and management

Each device should support multiple keys. For example, a device manufacturer might want to use a key to protect the bootloader so that only authorized and encrypted code will be loaded.

A system integrator who puts devices from multiple manufacturers into a CAN system should also be able to generate and save keys. These keys would pair the installed devices (potentially from different manufacturers) and not allow new devices to be introduced without authorization by the system integrator.

Last but not least, on the user level, it might be desirable to generate temporary keys for plug-and-play devices. A user might have authorized a particular plug-and-play device but does not want to allow additional devices or replacements without authorization.

### 2.3.1 Key storage in the devices

In the participating devices, permanent keys or the last session key need to be stored in non-volatile memory. Depending on the device and how it implements the storage, an intruder might try to get access to this non-volatile memory to get access to the keys.

To illustrate the vulnerability, imagine the device is Linux based and has a file system. Storing the plain keys using the file system would be simple but also an easy target. If the device also has Internet access and ever gets hacked, reading the keys from the file system becomes easy.

A device that does not use an operating system is more difficult to hack. But attackers have shown that they can load their own code if a built-in bootloader is not well protected. If the keys are just stored in a connected EEPROM, an intruder might be able to read the key.

We should make it reasonably difficult for an intruder who hacked a device to get access to the stored keys. These are some precautions we can take:

- the key storage location should not be obvious
    - if a file system is used, do not name file “keys”
    - if EEPROM is used, do not store keys at the beginning or end
    - offsets/filenames to the keys should not be constants; generate the keys dynamically as part of your initialization code

- hide the key with random data
    - if you can spare the memory, place the key within a bigger random data block
- encrypt the keys saved
    - do not store the keys as a copy;
      instead use some minimal encryption method on them

These methods do not add 100% security but raise the difficulty level for potential intruders to get easy access to the stored keys.

### 2.3.2 Key storage outside of the CAN system

Each time a key is generated, we need to ask ourselves if the key also needs to be stored outside the system. Potentially this creates the need to maintain a database with all keys ever generated. And that makes a very interesting target for attackers. If this option is chosen, the key copies stored need to be well protected by other security means.

#### *Use of security "dongles"*

One option for increased security could be that such keys are not stored on any PC, but only in hand-held security devices or "dongles". Only if you have physical access to one of these dongles can you make changes to the security settings of a device or system. Dongles can be based on existing CAN/CANopen handheld diagnostic tools such as CANopen Diag. A cloning function allows creating backups or copies of a dongle. The drawback is that now all keys (or a group of keys) is in one physical device, but on the plus side the keys are never stored anywhere on the Internet.


# 3 CANcrypt functionality

The first proof-of-concept implementations of CANcrypt were done on multiple NXP LPC17xx devices and a PC with a PEAK PCAN driver interface. Demo code is available for download at www.esacademy.com/cancrypt.

## 3.1 Summary

With CANcrypt, we offer a framework to handle both authentication and encryption of CAN messages. As there is some message overhead, the CANcrypt security features should be used only by a limited number of devices (the current version supports up to 15 devices) and only for selected messages (selected by CAN message ID). Depending on the chosen security level, encryption may be used not only on entire messages but also on selected bytes.

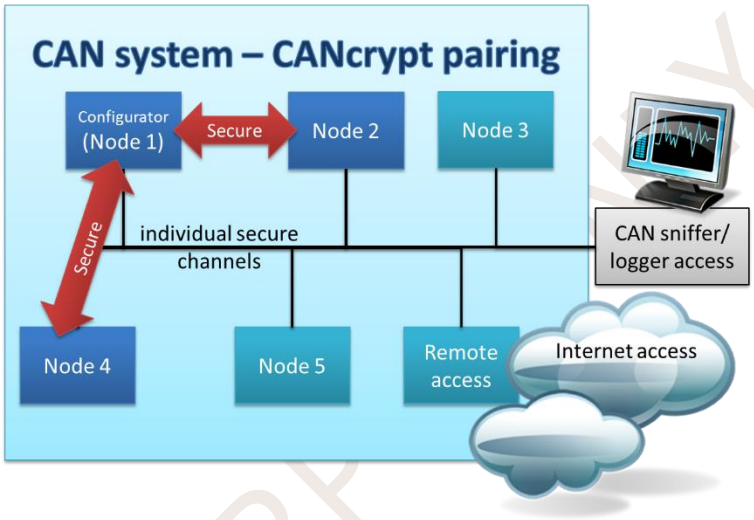
Security features are based on shared symmetric keys. There is a group key for all devices participating in the secure communication and a pairing key for secure channels between two devices. The secure pairing channel has a higher security level for use in system configuration or especially sensitive point-to-point connections such as bootloader communication.

### 3.1.1 Pairing

The CANcrypt pairing mode connects a CANcrypt configurator with a CANcrypt device and provides a secure communication channel supporting both authentication and encryption.

Secure messages are transmitted in pairs, first a preamble message that contains security configuration details and a signature followed by the message with the data.

The dynamic pairing key used between paired devices is continuously updated by introducing new bits generated as described in section (2.2.1 “The bit-generation cycle”). The update frequency is configurable.




SECURE CHANNELS IN A CAN SYSTEM

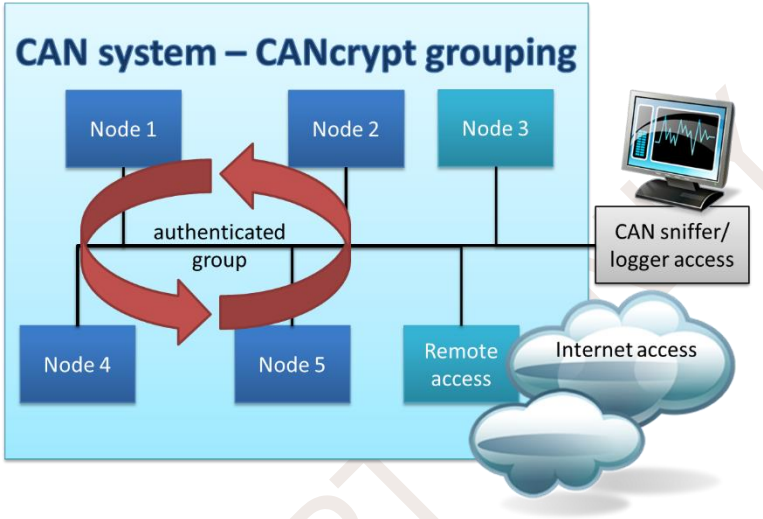
### 3.1.2 Grouping

The CANcrypt grouping mode establishes a group of secure devices. In this mode, every device produces a secure heartbeat. The dynamic grouping key is updated based on random values in the heartbeats. No other messages use security features.

All grouped devices monitor the network for manipulations (injections, collisions in the data field) and stop producing the secure heartbeat on detecting such a manipulation.

Receiving a secure heartbeat indicates that all previous messages from the transmitting device are authentic – otherwise the device would not have produced the secure heartbeat.

Note: due to application specific delays in drivers and buffers it might be necessary to wait for two following secure heartbeats before considering a message authenticated.




AUTHENTICATED GROUPING IN A CAN SYSTEM

## 3.2 Basic functionality

In this section, we outline the basic functionality provided by CANcrypt. This includes generation and updates of keys, generation of the one-time pad, and the generation and evaluation of the secure message pair.

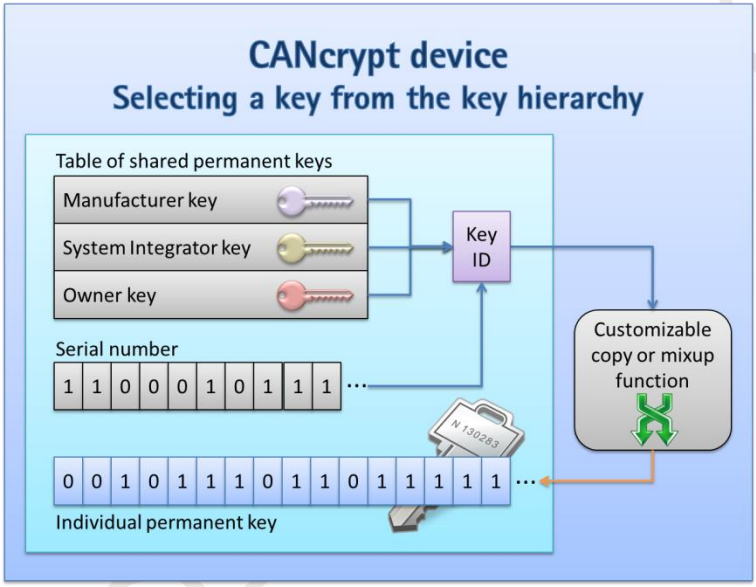
### 3.2.1 Key management and key hierarchy

Security systems require keys. Security keys require management. Who keeps a copy of which key where? Does a manufacturer need to keep a copy of each individual key of every product ever produced? Which keys does a system builder or integrator need access to?

To support multiple keys at different security levels (for example for the manufacturer, system integrator, and owner of a system), CANcrypt implements a key hierarchy of up to six keys. Each of these keys has a key ID, and the higher the value for a key ID, the higher the security level.

Keys can never be read from a CANcrypt device. They can only be erased or newly generated. To erase a key, a configurator must establish a direct secure connection (active pairing) to a single device based on one of the stored keys. Once the devices are paired, the configurator can erase keys of the same or lower hierarchy level only.

In summary: once a key is generated and saved, it can only be erased and re-generated if paired based on a key of the same or higher security level.



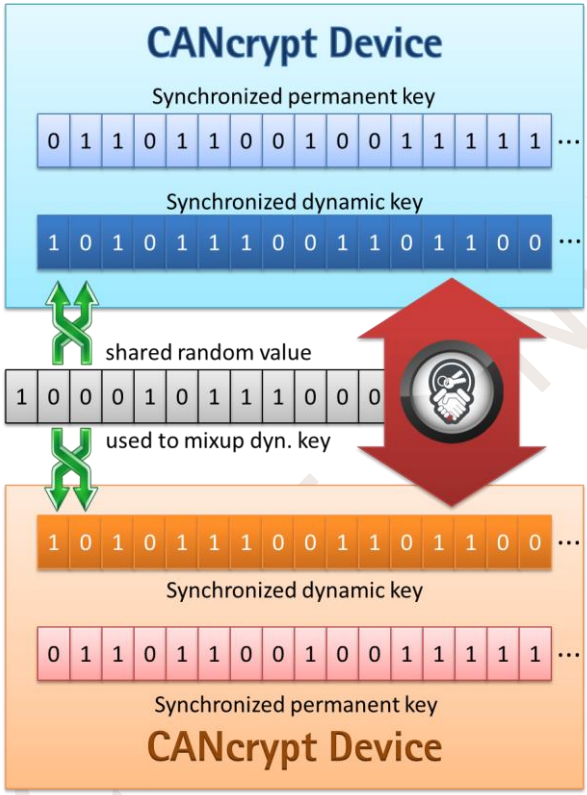

KEY SELECTION FROM KEY HIERARCHY

The pairing process requires one permanent key and may also involve an optional serial number as illustrated in the figure above, “Key selection from key hierarchy”. This method allows a manufacturer to use the same base key in multiple devices. As pairing (establishing a secure channel) may also involve the serial number, a service or maintenance login could still be device specific.

### 3.2.2 Updating the shared dynamic keys

The dynamic key gets continuously updated following a fixed time scheme. Depending on the configuration, typical update cycle times are 500 ms, 1 s, or 2 s.

For a single pair of devices, a single new bit is generated randomly, imitated by the configurator. With multiple devices, the secure heartbeat is used to introduce new random values to by all participants.

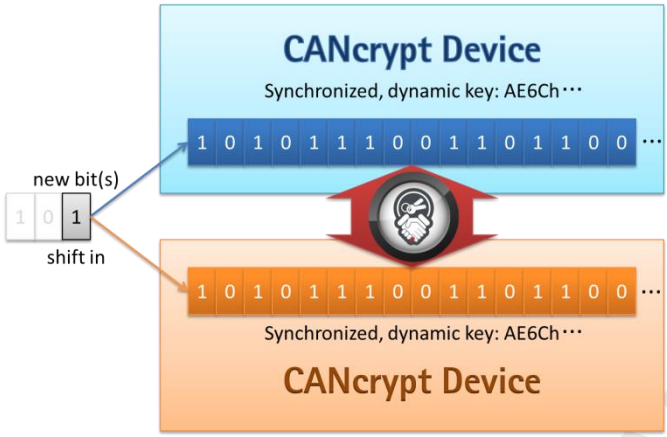



DYNAMIC KEY GENERATION WITH SHARED RANDOM NUMBERS

As part of the secure heartbeat, all participating (grouped) devices exchange encrypted random numbers. These shared random numbers are used to generate a new synchronized shared key as illustrated in the figure above. Up to 15 devices can actively participate in this mechanism.

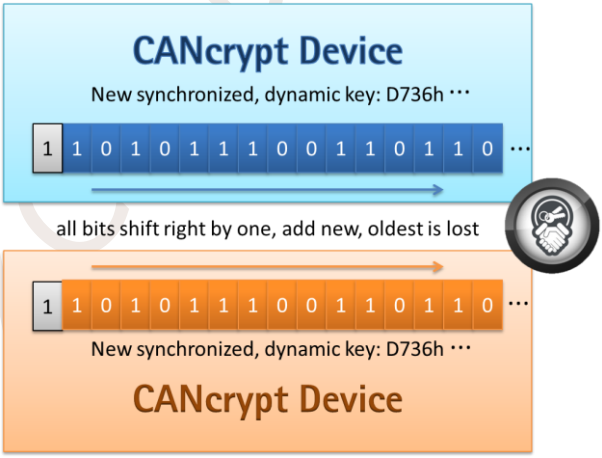
This dynamic key is re-generated with every secure heartbeat cycle.

In paired mode (only two devices involved), the random-bit-generation cycle is used to introduce new bits to the shared dynamic key.




ADDING A NEW BIT TO THE DYNAMIC KEY

The new bit or bits get shifted into the dynamic key (shift right). This is done in parallel by both paired devices as illustrated in the figure above, “Adding a new bit to the dynamic key”. The figure below, “New bit is shifted in”, shows the new dynamic key now used by the devices. This updated key is now used for future pseudo one-time pad generations until a new bit gets introduced.




NEW BIT IS SHIFTED IN

### *Synchronization challenge*

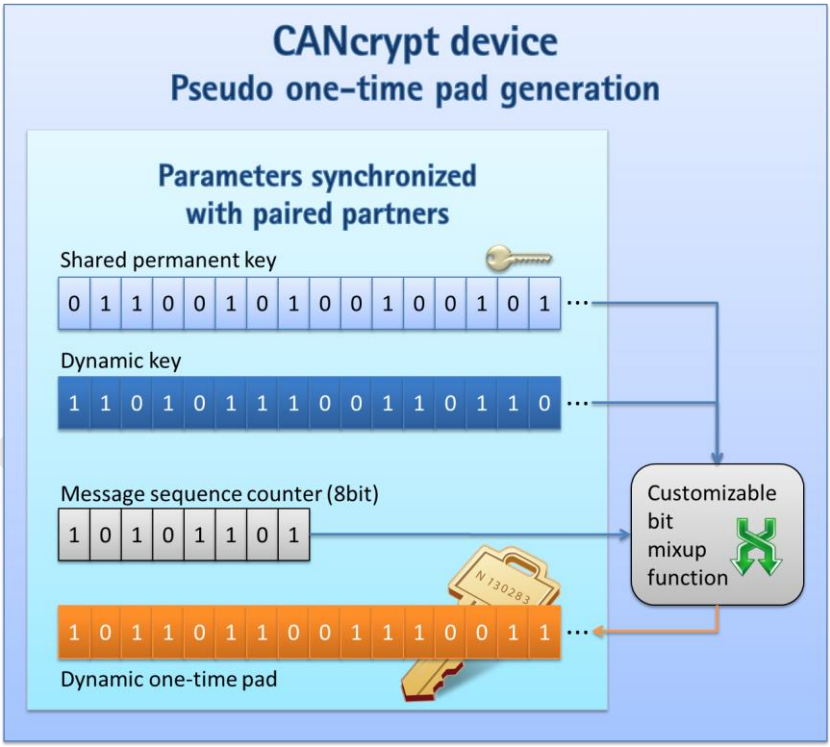
Even if the key update is executed by all CANcrypt devices in parallel, a secure message might still be received using the previous key. Therefore all devices must

keep a copy of the previous dynamic key to decrypt and authorize messages that still use the previous key until the key update has been executed by all nodes.

### 3.2.3 One-time pad generation

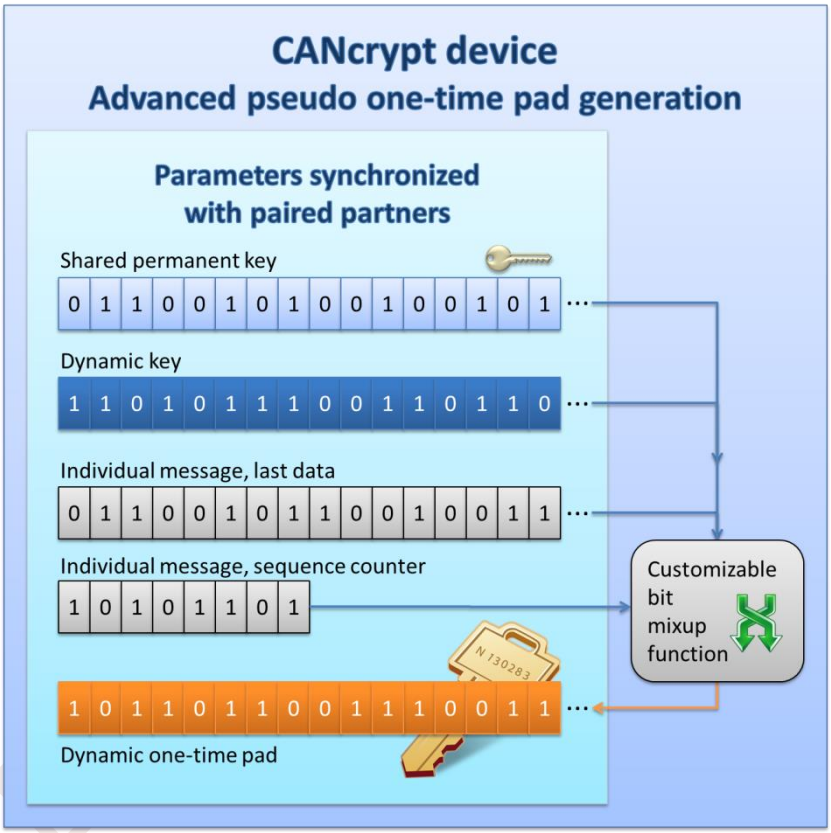
Besides the shared dynamic key, devices also share the permanent key and a message counter (not secret) as illustrated in the figure below, “Shared parameters for pseudo one-time pad generation”. The message counter is part of every secure message pair and is transmitted with the preamble message.

The dynamic one-time pad is regenerated with each transmit or receive of a secured message. The value is based on the current dynamic key, but the bits are rotated and mixed depending on a combination of the current transmit message counter and the permanent key. This method ensures that the dynamic one-time pad’s bits experience a significant change between each use. Each device needs to maintain two message counters, one for transmit and one for receive, to be able to create the corresponding dynamic one-time pad.



### Shared parameters for pseudo one-time pad generation

In an advanced custom version of CANcrypt additional inputs can be used for the generation of the one-time pad. This can involve decrypted data from previously received messages, for example from the secure heartbeats. Instead of light-weight Speck bit mixup function, the more advanced AES-128 or AES-256 algorithm can be used to create the one-time pad.



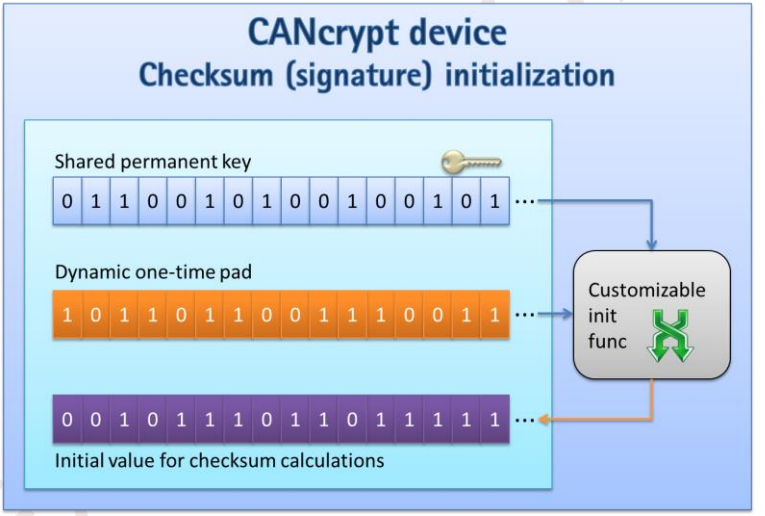

Advanced one-time pad generation

### 3.2.4 Generating an initializer for the CANcrypt checksum

CANcrypt uses checksums for the secure message table (containing configuration data), the secure heartbeat and secure messages. The checksum for the message table is calculated with an initializer of FFFFh.

As the other checksums only cover a limited number of bytes, they must not use a simple initializer like zero or FFFFh. The default CANcrypt configuration is that the initializer for these checksums is taken from a combination of the permanent key and the dynamic one-time pad. This ensures that the initializer varies from message to message.

For advanced use, the function generating the initializer can be customized, or a completely different scheme like a security hash function or a safety protocol usable checksum may be implemented.

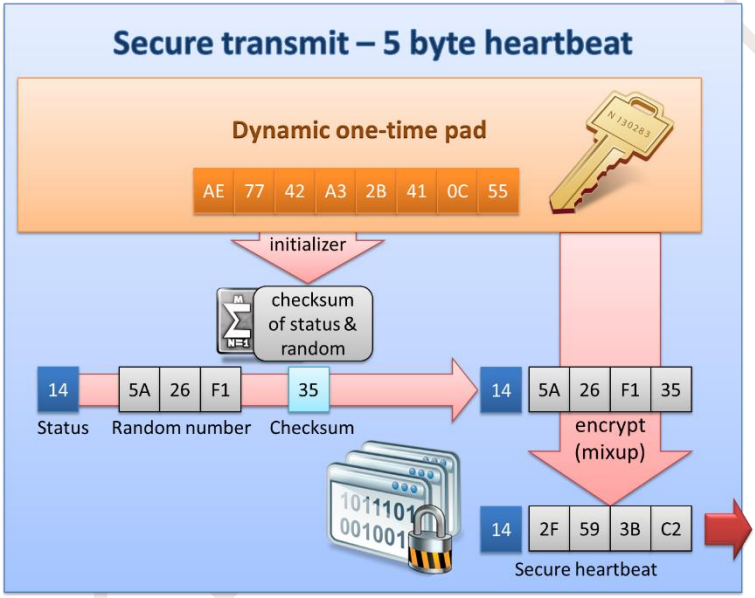



GENERATION OF CANCRYPT CHECKSUM INITIALIZER

### 3.2.5 Transmitting the CANcrypt Secure Heartbeat

The secure heartbeat consists of five bytes. The first byte is the heartbeat status byte indicating if the device is actively paired and the communication is still authenticated. The following three bytes are random numbers. The last byte is a checksum of the status byte and the three random bytes.

Before transmitting, the last four bytes (random number and checksum) are encrypted based on the current dynamic key.




SECURE HEARTBEAT GENERATION

The timing for the secure heartbeat is controlled by the detection of other secure heartbeats and an event and an inhibit time as defined in CANopen.

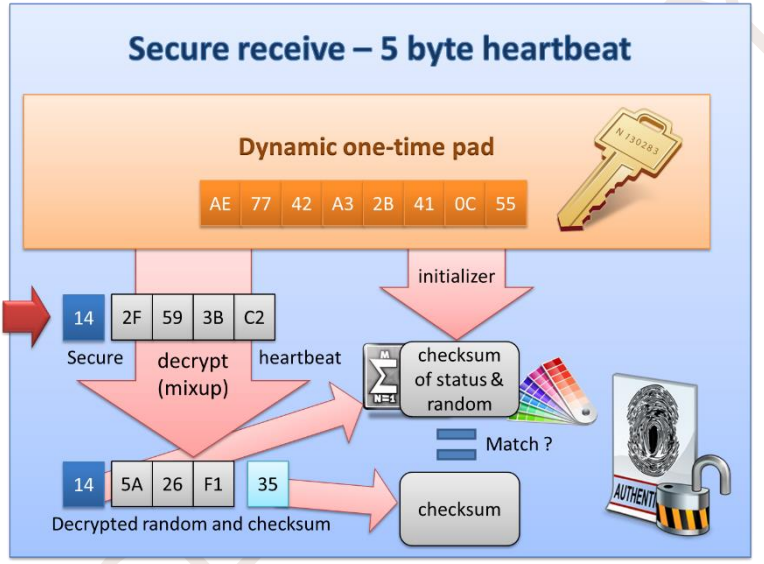
On receiving a secure heartbeat, a CANcrypt device participates in the heartbeat cycle by transmitting its own secure heartbeat and resetting its internal timer. Any device detecting an expiration of the event time starts the next secure heartbeat cycle by transmitting its own secure heartbeat.

Any device may start the next heartbeat cycle earlier. For example, a device might want to have a received message authenticated as fast as possible. However, the

device must wait until the inhibit time has passed before initiating a new cycle. This delay ensures that the CAN bus is not flooded with heartbeat cycles.

### 3.2.6 Receiving a CANcrypt Secure Heartbeat

On receiving a secure heartbeat, a device first decrypts the last four bytes based on the current shared dynamic key. Then the device calculates the checksum of the status byte and the random bytes. If the calculated checksum matches the transmitted checksum, the heartbeat is considered authenticated.

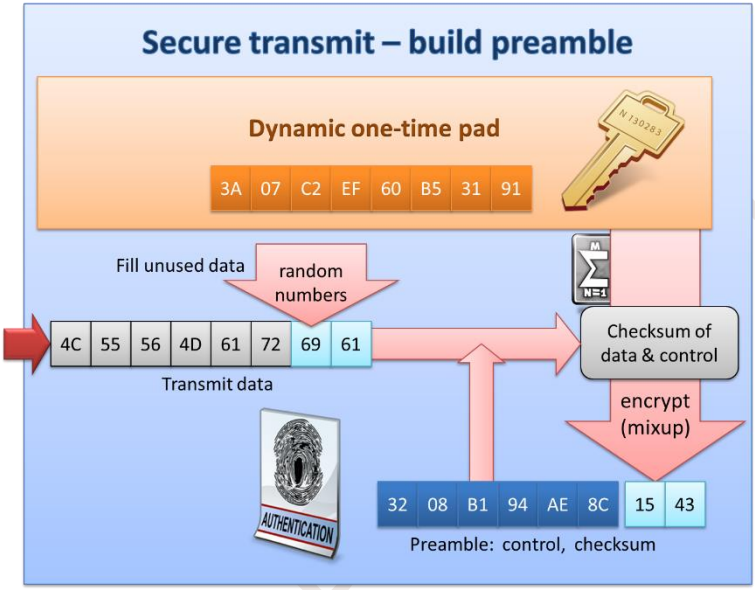



SECURE HEARTBEAT VERIFICATION

On receiving a secure heartbeat, a device determines if the local inhibit time has expired – if the time since the last transmission is greater than the inhibit time. If so, the device transmits its own next secure heartbeat.

### 3.2.7 Transmitting a secured message

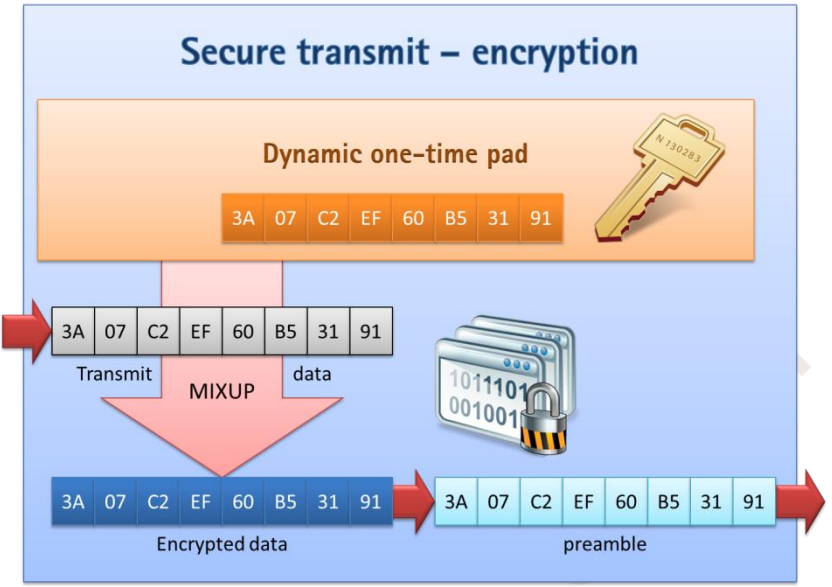
The secure transmit handler checks if a message to be transmitted is in the global configuration list for secure messages.




GENERATING THE PREAMBLE

If the message to be transmitted is in the list of secure messages, a preamble message is generated. The message contains control bytes including the CAN message ID to follow and the current transmit message counter.

Then the 16-bit signature is generated by calculating a checksum and encrypting it (the method is configurable; the default is an exclusive OR). Encryption happens based on the dynamic pseudo one-time pad.

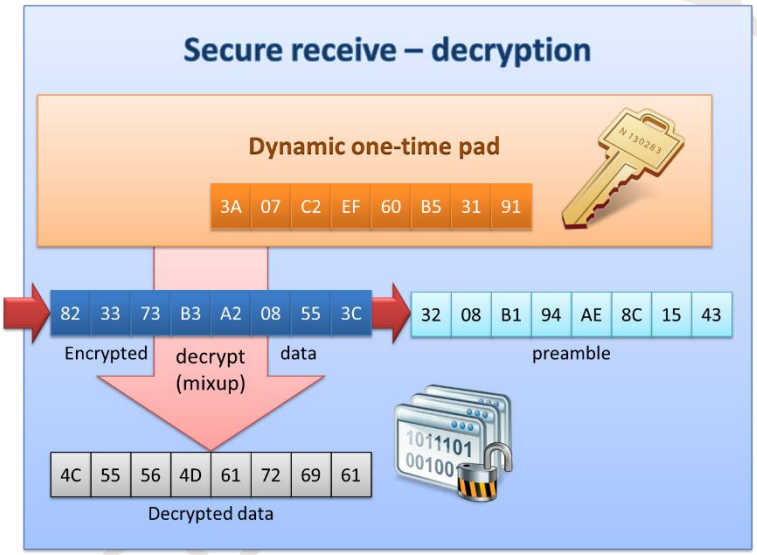



ENCRYPTING TRANSMIT DATA

If encryption for the data is used, then the data bytes requiring it are encrypted (configurable, default is exclusive OR), also based on the pseudo one-time pad. Both messages are transmitted back-to-back on the CAN system.

### 3.2.8 Receiving a secured message

On the receiving side, if a message is received that is listed in the secure message list, the preamble is received first and stored in a buffer. The reception starts a 10 ms timeout. A preamble that is received without a message following within 10 ms is considered an error, and a security error counter gets incremented. The included message sequence counter is checked. The counter contains information about the dynamic update cycle and can be used to determine if the current (latest, newest) dynamic key or the previous one gets used.

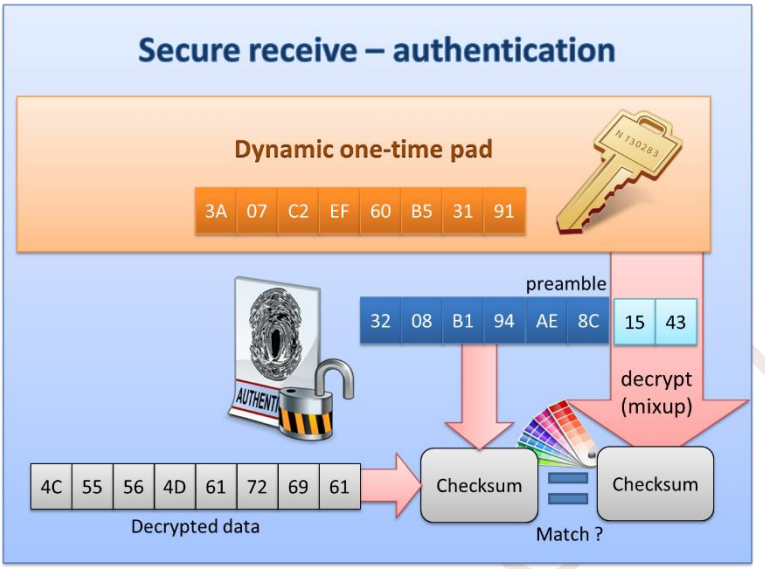



RECEIVING PREAMBLE AND SECURED MESSAGE

Once the secured message is received, the local pseudo one-time pad is generated using the message sequence counter from the preamble. If parts of the message are encrypted, they are now decrypted.

Next, the signature needs to be verified. To do that, the checksum is rebuilt on the preamble controls and the message data. The signature received with the preamble is decrypted and the two are compared. If they match, the message is considered authenticated.

A secure message received is passed on to the application or protocols above the CANcrypt handler only if authentication was successful. Otherwise the message does not get passed on and is “invisible” to a device.




AUTHENTICATION BY RECEIVER

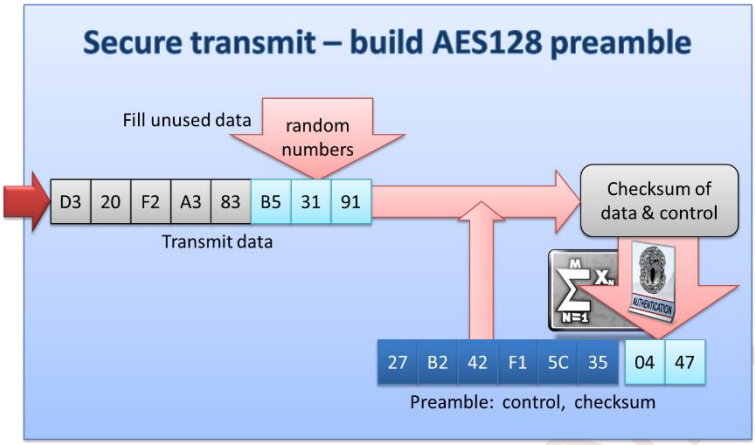
### 3.2.9 Multi message stream with epilog message

There might be situations where larger data blocks are transmitted with back-to-back messages. To optimize these transfers (by not requiring a preamble with every message), CANcrypt supports message sequence handling for up to eight messages.

If the control/request byte in the preamble specifies that multiple messages are part of the sequence, the signature of the preamble is not used. Instead an epilog message is inserted at the end of the sequence. The format of the epilog is identical to the preamble and contains the signature for all messages transferred with the sequence. See section **Error! Reference source not found.Error! Reference source not found.** for details about the message stream.

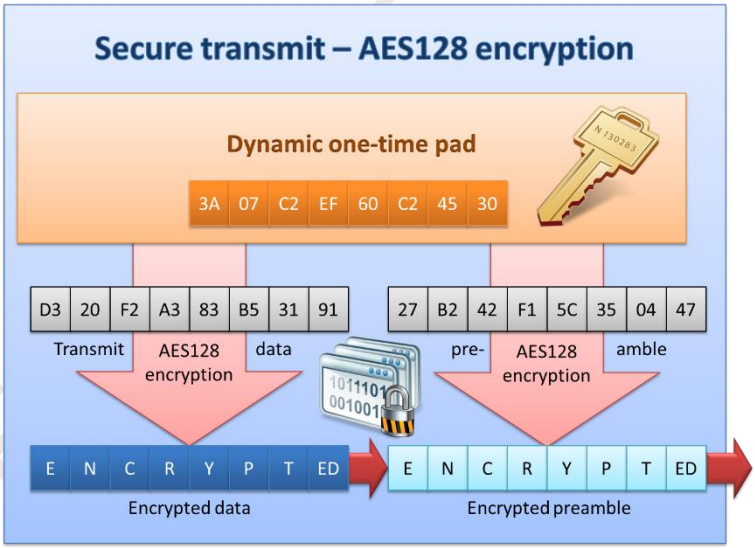
### 3.2.10 Transmitting an advanced secured message

In advanced mode, the generation of a secure message can be based on AES-128 encryption. As unused bytes are filled with random data, the total message size (consisting of the preamble and data message) is exactly 128 bits. The following figures illustrate the processes involved for encryption and decryption.



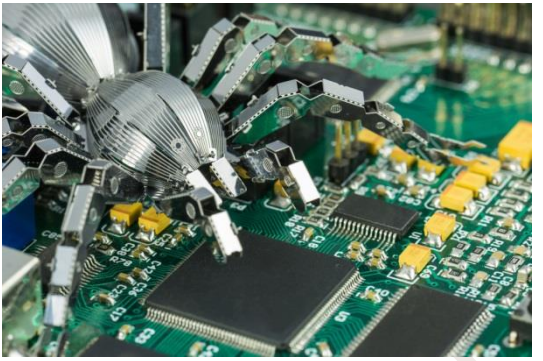

GENERATING THE PREAMBLE – AES-128 VERSION

Unused bytes in the data message are filled with random values. The preamble and data message are each eight bytes.




ENCRYPTING TRANSMIT DATA – AES-128 VERSION

Then both the preamble and data message are AES-128 encrypted using the current dynamic one-time pad.



# 4 CANcrypt attack vectors

In this chapter we examine the remaining attack vectors of CANcrypt.

## 4.1 Randomness

Wherever random numbers are used, a typical attack vector is to assume that the numbers are not random and to determine a pattern. The random numbers used by the paired devices must be “reasonably good”, and they must differ with every power cycle.

This might be a challenge for many microcontroller systems. The random functions provided by C compilers are typically not good enough to ensure “randomness”. If these are used, it is important to update the random seed used by this function as often as possible.

One possible option is to maintain and update a seed value that is constantly updated (using any arithmetic function) depending on inputs from A/D converters (especially highest resolution bits), high-resolution timers (measuring timestamps of CAN messages received).

Latest microcontrollers more and more often have a dedicated random number generator. Where available, these should be used.

## 4.2 Creation and storing of permanent keys

The phase where keys are stored permanently is crucial. Even if an intruder cannot see keys generated by the CANcrypt methods, a potential intruder present at this stage would be a security concern. So whenever initial pairing keys (no matter

if by the manufacturer or system integrator) are generated and stored, extra precautions should be taken to ensure that no intruder is present.

The precautions can include verifying that only those devices that are physically connected to CAN are required for this process.

Some microcontrollers offer dedicated memory locations for storage of security information. Where available, this area should be used to store the key(s) with the highest priority.

If “regular” FLASH memory is used, ensure that the FLASH protection bits are set. Specific functionality of these depends on the manufacturer. Usually these can be configured in a way that only the code executed from the FLASH memory can read-access this memory. Where possible, protect the FLASH from being read from RAM and FLASH programming ports/interfaces.

## 4.3 Debug and bootloader interfaces

Microcontrollers with an enabled JTAG or SWD interface for debugging or an internal bootloader to reprogram the device can be very vulnerable to attacks. This is especially true when these accesses are also made available through other communication interfaces such as UART, USB, or even CAN. Using an appropriate debugger or programming tool, an intruder can manipulate memory contents and inject code.

For highest security, all such interfaces should be internally disabled. Bootloaders should only be enabled if they offer their own security levels such as protection from unauthorized activation and supporting flashing only of authorized files.

## 4.4 Read/write access to CAN system

This section summarizes attacks where the intruder has full CAN level access and can receive all messages and inject messages at will. The access could be through a physically connected sniffer or a remote access device.

In CAN networks, a typical attack involves recording messages and replaying them. If the messages exchanged after power up are always the same, an attacker could fake initial messages by replaying them. However, due to the dynamic, random key changes, a hacker would probably look at alternate methods first. Examples include trying to activate a boot loader or re-flashing a device with new code.

## 4.5 Ability to physically remove/replace devices

With physical removal, an attacker has the ability to reprogram (re-flash) a connected device.

If one of the paired or grouped devices is removed and replaced with a tampered device, the tampered device can participate in the key generation or grouping cycles. However, all pairing and grouping functions also use elements of a permanent key. With endless time and re-tries an attacker might be able to determine parts of the permanent key. To make this attack vector less attractive, CANcrypt uses incremental delays on pairing/grouping re-tries.

## 4.6 Signal level access to CAN and PCBs

The result that can be achieved with signal-level access heavily depends on many factors. With PCB access to a CAN transceiver, an attacker would be able to see the bit generation. If session pairing is used, the attacker can see all keys generated. If permanent pairing is used, an attacker would still be able to learn changes to the dynamic key over time and eventually have a copy of the dynamic key. Although this level of attack is theoretically possible, any active attack trying to manipulate or fake data would still be difficult as it would require hard, real-time reaction.

Note that this attack option is not available if the paired devices use controllers with integrated transceivers such as some of the NXP LPC11Cxx devices.

## 4.7 Summary

At this point we are not aware of any “promising” attack vectors on the CAN/CANopen level in a CANCrypt system. That includes remote access (for example through a hacked gateway) as well as direct access with a CAN sniffer utility.