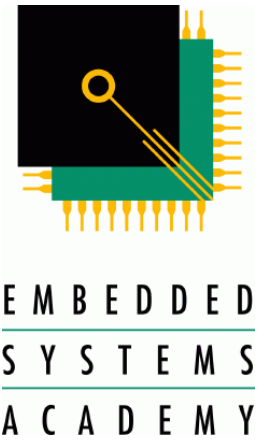
Secure CANcrypt Bootloader for NXP LPC546xx

for version 1.0



Published by
Embedded Systems Academy
1250 Oakmead Parkway, Suite 210
Sunnyvale, CA 94085, USA
www.esacademy.com

CANcrypt technology from
"Implementing scalable CAN security with CANcrypt"
ISBN 978-0-9987454-0-4
www.cancrypt.eu

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 with this manual 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 manual 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.

Contents

1 Bootloader functionality and security

1.1 Outline

The software described in this manual implements a secondary bootloader for the NXP LPC546xx microcontroller family. It uses the CAN FD interface and uses protocols and mechanisms from CANcrypt and inspired by CANopen. A binary version of the bootloader implementing a default configuration is provided at no cost (.hex download from NXP web pages). The commercial version including all source files allowing numerous configuration options is available from Embedded Systems Academy.

The encryption and authentication methods used for CANcrypt pairing and the file containing the code to be programmed are held in separate modules to allow an easy exchange.

- The default method for CAN based authentication is CANcrypt pairing at “regular” security level based on CAN secret bit generation and the Speck cipher.
- The default method for code update file encryption and authentication is AES-GCM.

Both default methods use an own 128bit symmetric key stored in regular Flash memory.

1.2 Security Limits and Risks

Security protection can never be “100%”, some limits and risks remain. Here we summarize the known limitations and risks.

1.2.1 Primary vs. secondary bootloader

This secure bootloader is a secondary bootloader. The NXP LPC546xx also has multiple primary, internal on-chip bootloaders. At any time, someone with physical access to the microcontroller can use the internal bootloader to erase the program memory or load any new code, unless they are disabled.

The NXP LPC546xx microcontroller family supports disabling all internal bootloaders and the debug interface (SWD). Be very cautious to do so. If all other means to re-program the device are disabled, a system can no longer be updated, if the key(s) to the secondary bootloader are lost!

1.2.2 Bootloader activation

Also, any application loaded must have a means to activate the bootloader. If the application fails to execute this functionality, devices might no longer be updated. The com-

mercial version of our secure bootloader has a configurable delay after power-up which can be used as a timeout window to “catch” the bootloader by sending it an activation message directly after reset.

1.2.3 General CAN vulnerability

On the CAN communication side, keep in mind that CAN is always vulnerable to denial-of-service style attacks. If a CAN interface is flooded with error messages or signals, the CAN interface shuts itself off.

1.2.4 Trustworthy production environment

The code protection key and possibly default CANcrypt connection key must be installed in a trustworthy environment of the manufacturer. This way this key is only known to the manufacturer of a device. Only the manufacturer can generate code files that are accepted by the bootloader.

The CANcrypt connection may also be installed at that time. Alternatively, this key is generated during an initial pairing of a host system with the bootloader. This could happen on a system integrator level. A system integrator building a system could execute the pairing upon initial system power up. In this case the system integrator is responsible for a trustworthy environment (protecting the connection key) for the first power up of a device.

1.2.5 Key storage

As with any security key based system, it is the responsibility of the manufacturer and the system integrator to protect their symmetric and possibly private asymmetric keys. On one hand, enough backups of the keys are needed, on the other hand every additional copy of the key increases the risk that an unauthorized person gets access to it.

If the number of keys that you have to save is limited, then one of the commonly available keyword safe (password manager) programs like KeePass can be used to generate and save keys.

1.2.6 Key selection – DO NOT USE DEFAULTS

The demos provided with this secure bootloader use default keys. Never use these for any real application, otherwise you might as well drop the use of security altogether.

As usual with keys and passwords, use random keys, not any pattern or names or birthdates. On any brute force attack, hackers will try such patterns first.

1.3 Security goals

In this section, we define our security goals.

1.3.1 Security methods and keys

Per default, this secure bootloader uses two symmetric keys to fulfill two security goals:

1. The secure bootloader ensures that only an authenticated communication partner can erase and load new code. The host (for example Flash Magic) connecting to the bootloader initiates the CANcrypt pairing process based on a shared symmetric ***CANcrypt connection key***.
2. The code file received from that host is AES-GCM encrypted and authenticated based on a shared symmetric ***code protection key***.
   Optionally, the commercial version supports use of different methods including different key length and asymmetric keys (public/private) used for RSA or EEC algorithms.

Optionally, the commercial version supports use of different security methods and different key lengths.

1.3.2 Multiple keys required

An attacker that has access to the CAN FD bus (e.g. some remote access to read/write CAN messages) will not be able to introduce malicious code to the system, unless he gains access to both the code protection and connection keys.

1.3.3 Key hierarchy

The commercial CANcrypt version allows enabling a key hierarchy. If enabled, the higher security level manufacturer key can be used to erase the system integrator key, which can then be newly generated.

NOTE: when this feature is enabled, the manufacturer key should not be identical to the code protection key, otherwise a single key has the power to both generate protected code AND pair a CANcrypt configurator with the bootloader.

1.4 Software components of the bootloader system

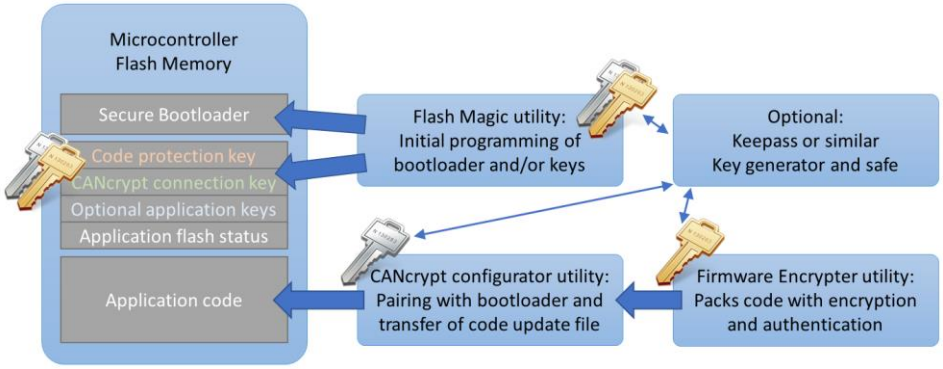


FIGURE – SOFTWARE COMPONENTS AND KEYS

The previous figure illustrates most components required to install and use the bootloader. The components in detail are

- Secure bootloader binary (.hex file, free for LPC546xx) or source code (commercial version from ESAcademy)
- Bootloader Post Processor utility to add bootloader checksum and encryption keys (manufacturer & system integrator) to the bootloader hex file
- Flash Magic utility to load bootloader with encryption keys into microcontroller and activate hardware security features (code read protection)
- Hexsum utility to convert a regular application hex file to a binary firmware update file with CRC checksum
- Firmware Encrypter utility to add encryption and authentication to the firmware update file
- CANcrypt configurator utility (basic functionality provided by Flash Magic) to securely pair with the bootloader and transmit code update file
- Optional key generator and key safe software, e.g. KeePass

1.4.1 Firmware update cycle

The figure illustrates the path of the new firmware code for an update. It gets generated at the manufacturer of the Embedded System and is encrypted using the code protection key, only known to the manufacturer.

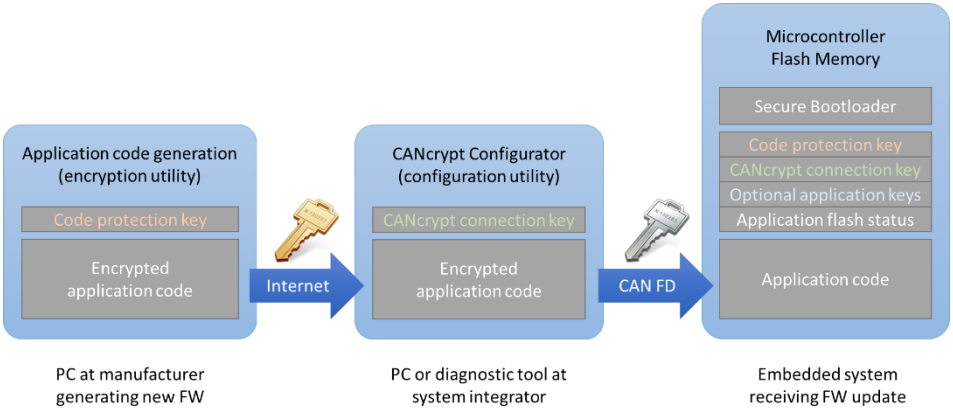


FIGURE – FIRMWARE UPDATE PATH

The encrypted code file is send to the system integrator or service technician who has access to the system integrator key. Although the code file is encrypted, a secure transfer method (e.g. VPN, SFTP) should be used to copy the code file to the diagnostic utility or PC of the system integrator.

This diagnostic tool or service utility uses secure CAN FD communication to pair with the embedded device and transfer the code file to the target device. For this connection, the CANcrypt connection key is required.

1.5 Deliverables

The ESAcademy secure bootloader for the NXP LPC54618 is delivered as packed directory. The sources to the bootloader are only available with the commercial version.

A PEAK PCAN-USB FD (www.peak-system.com) or PCAN-USB Pro FD interface is required to communicate with the secure CAN-FD bootloader. The programming process can be executed with the Flash Magic utility available at www.flashmagictool.com. For monitoring the CAN-FD communication we recommend CANopen Magic ultimate (www.canopenmagic.com), which also offers CANcrypt interpretation of messages transferred.

Directory "bootloader"

Contains the bootloader as hex file and the BootloaderPostProcessor utlitiy to generate a bootloader configuration. The file "secure_bootloader_x_x_defaultkeys_demoonly.hex" is a bootloader version using default keys.

Directory "doc"

Contains this manual and release notes.

Directory "sampleapp"

Contains an example application that can be loaded using the secure bootloader. The file that can be directly used with the default keys is "lpcxpresso54618_test_app_sec.bin".

The entire sources to the application are provided in a zip archive.

Directory "utilities"

Contains the "BootloaderPostProcessor.exe" and the "FirmwareEncrypter.exe".

The BootloaderPostProcessor is used to patch the bootloader hex file with the code protection and the CANcrypt connection keys.

The FirmwareEncrypter is used to pack an application into a secured code update file.

2 First setup and step by step demo

The demo was tested on the NXP LPCXpresso546xx Eval Board Rev C equipped with the NXP LPC54618 microcontroller and the CAN-FD Shield Rev C add on module.

Observe

The numbering of jumpers on main board and CAN-FD shield overlap. In this manual we refer to jumpers "on main board" and jumpers on "CAN-FD Shield".

To operate the secure CAN-FD bootloader you require the latest version of Flash Magic (www.flashmagictool.com) and a PEAK PCAN-USB FD (www.peak-system.com) or PCAN-USB Pro FD interface.

2.2 Hardware and cabling

Main board setup

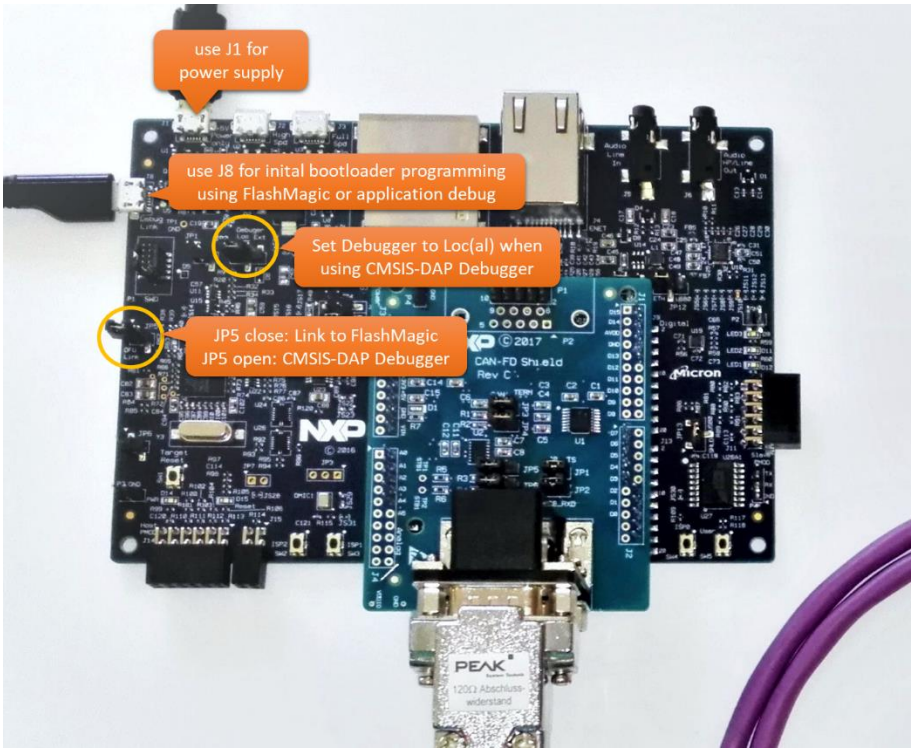


FIGURE – BOARD SETUP AND CONFIGURATION

Power the main board using J1. Power supplied via the debug port may not be sufficient to ensure good CAN voltage levels and can cause error frames.

When using J8 Debug Link (CMSIS-DAP Debugger) for debugging your application, set JP2 to “LOC” and remove JP5 DFU Link.

When using Flash Magic to program the initial bootloader code, close J5.

CAN-FD Shield and cabling setup

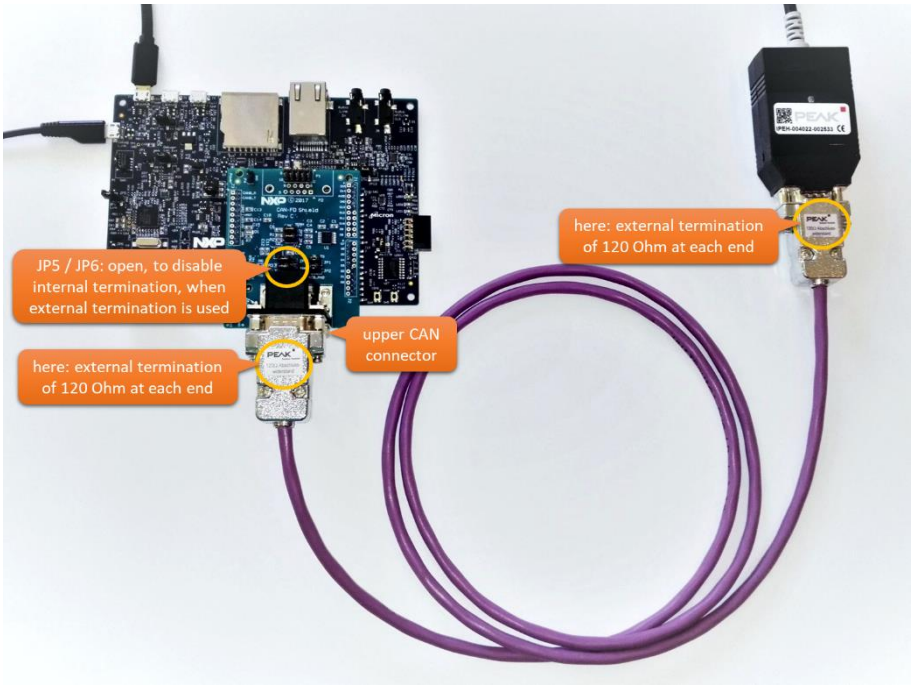


FIGURE – CAN CABLING AND SETUP

The bootloader uses the CAN 0 port of the CAN-FD Shield (upper connector).

For CAN-FD good cabling and proper termination is important. 120 Ohms termination are required on both ends of the cabling. The CAN-FD Shield has termination build-in, so if your cabling is not terminated, set Jumpers J5 and J6 on the CAN FD Shield.

Monitoring CAN-FD communication

While FlashMagic uses a PCAN-USB FD channel, the same channel cannot be used by PCAN View or CANopen Magic to monitor the communication. This means that either a second PCAN-USB FD or the second channel of a PACN-USB Pro FD is required to monitor the communication at the same time.

When you configure other CAN-FD devices (such as a monitor / analyzer), ensure that the bit timing and sample points are set as recommended by the CiA:

- Nominal Bit Rate 500kbps, 80% Sample point
- Data Bit Rate 2000kbps, 75% Sample point

When using PEAK utilities to configure a network, use the following parameters if the CAN clock is set to 40 Mhz:

Rate	Prescale	Tseg1	Tseg2	SJW	Sample
500kbps	1	63	16	16	80%
2Mbps	1	14	5	5	75%

2.3 Using the pre-generated demo files

NOTE: only use this bootloader version for a first test, it uses public known default keys and therefore offers no security protection. To enable security protection, enter your own keys to the key_xxx.txt files.

2.3.1 Programming the bootloader and default keys

Use Flash Magic to program the default bootloader as follows.

Ensure board is without power.

Set J5 to activate Link for Flash Magic.

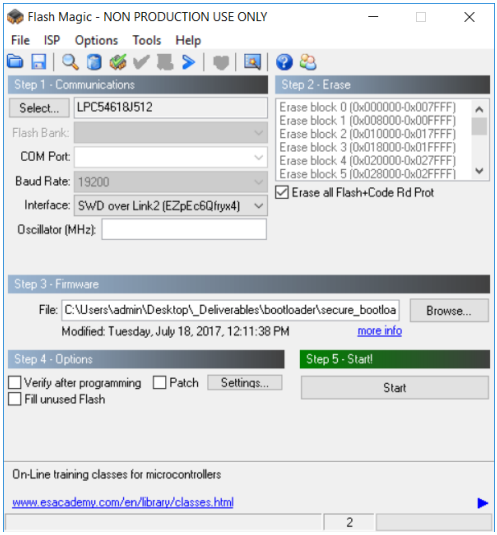
Put board under power (J1) and J8 USB cable connected to the PC on which FlashMagic is installed.

Start Flash Magic and in main FlashMagic window select

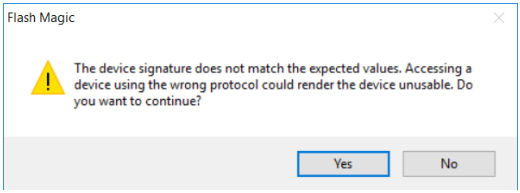
- Device: LPC54618J512
- Interface: “SWD over Linkxxx”
  (might require re-start of FlashMagic to detect)
- Checkmark “Erase all Flash+Code Rd Prot”

And select the firmware file
“secure_bootloader_x_x_defaultkeys_x.hex”

Hit “Start” to program the bootloader.



Depending on which specific device is present on the board, FlashMagic might recognize a mismatch of the device ID. Select “Yes” to continue.



To enable the bootloader, power cycle the board (reset is not sufficient in all cases). If you have a CAN-FD monitor connected (set to CAN-FD bitrate 500/2000), you will see a CANopen style bootup message with CAN ID 70Fh and one byte, zero. The message is repeated as heartbeat every second with the data byte 7Fh (stands for CANopen state pre-operational)

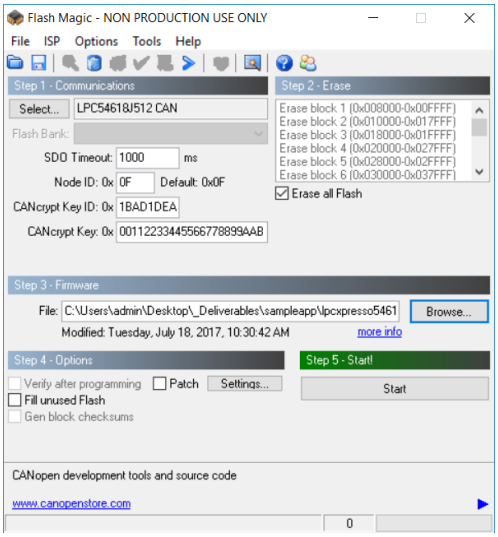
2.3.2 Using Flash Magic to load an application

You can now use Flash Magic to load an application.

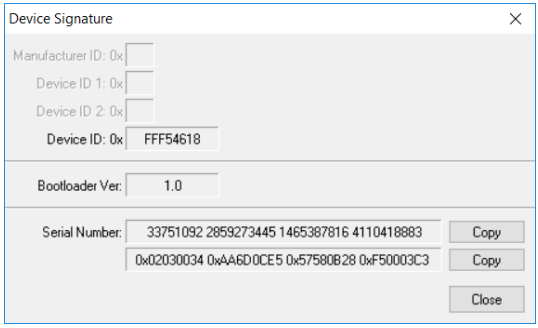
In main FlashMagic window select

- Device: LPC54618J512 CAN
- SDO Timeout: 1000
- Node ID: 0F
- CANcrypt Key ID: 1BAD1DEA
- CANcrypt Key: 00112233445566778899AABBCCDDEEFF

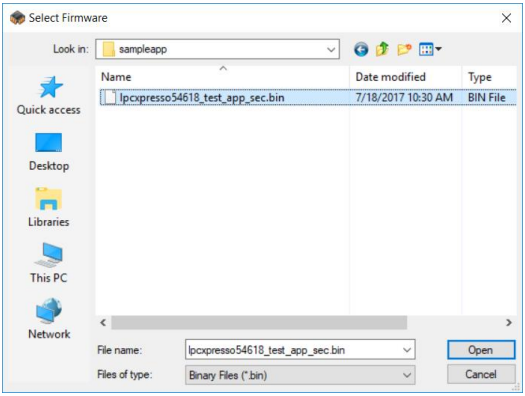
The CANcrypt key required here is the system integrator key programmed into the boot-loader.



Select menu ISP / Read Device Signature to ensure keys entered match and CANcrypt communication works.



Now select the firmware file “sampleapp/lpcxpresso54618_test_app_sec.bin” – note that default search is for .hex files only, switch to “all” or “bin”.



Hit “Start” to program the bootloader.

To activate the downloaded code, press the reset button on the main board. The LEDs are now showing a blinking pattern.

FIGURE – BUTTONS AND LEDS USED

To stop the application and re-activate the bootloader, press the SW5 button on the main board.

Implementation Note

Any application programmed must offer some means to re-activate the bootloader.

2.4 Use your customized bootloader and application

Proceed as follows to run the programming cycle based on your own bootloader configuration and application.

2.4.1 Create customized bootloader

First, configure the bootloader to use your own keys. The directory “bootloader” contains an example script file “generate_bootloader.cmd” that patches the two keys (manufacturer key for code protection and system integrator key for CANcrypt connection) to the bootloader code.

To use your own keys, put the desired keys in the text files “key_mf.txt” and “key_sys.txt”. Note that the key ID must also be changed, it is used as a unique public identifier (this number is returned by CANcrypt when asked which key is currently used).

Then execute the “generate_bootloader.cmd” command file to generate your “.hex” file with the bootloader and keys.

See section 3.3 for further details on the BootloaderPostProcessor utility.

If you purchased the source code version, various other configurations are available including different CAN bit rates, delays and timeouts.

2.4.2 Program your bootloader

This is the same as step 2.3.1 Programming the bootloader and default keys. The only difference is that now you do not program the default bootloader file but your customized bootloader file generated in the previous 2.4.1 section.

2.4.3 Add bootloader support to your application

Now generate your own application and secure code update file. The provided sample application in directory “sampleapp” can easily be modified to use a different manufacturer (code protection) key.

If you do not have MCUXpresso installed on your PC, proceed as follows:

- Unpack the archive “lpcxpresso54618_xxx.zip”
- Copy the desired key to file “key_mf.txt”
- Execute “Utilities/FirmwareEncrypt.cmd”

This generates the secure code update file Release/lpcxpresso54618_xxx_sec.bin

If you have MCUXpresso, import the project lpcxresso54618_test_app_mcuxpresso.zip into your MCUXpresso workspace (Import - General - Existing Projects into Workspace -

Next - Select archive file). Now build the 'Release' target. Note the call of the 'hexsum' and 'FirmwareEncrypter' tools in the post-build steps. The secure firmware update file is now at “lpcxpresso54618_test_app\Release\lpcxpresso54618_test_app_sec.bin”.

The manufacturer key is taken from the lpcxpresso54618_test_app\key_mf.txt file. Ensure that this key matches the key file in your bootloader.

See chapter 4 for more details on creating your application, also in regards to bootloader activation.

2.4.4 Program your application

This is the same step as in section 2.4.4 Using Flash Magic to load an application. The only differences are that now you need to enter (copy/paste) the key ID and key you used as system integrator key (from file “key_sys.txt”) when creating your customized bootloader and the update file to select is of course the one you generated in the previous step 2.4.3 and not the default file.

3 Bootloader configuration

3.1 Default Settings

The bootloader is based on a CANcrypt implementation with

- default bitrate of 500 kbps nominal and 2000 kbps data rate for CAN FD
- default CANcrypt device ID (and CANopen ID) is 15
- no power-up delay (to offer a back-up option – time delay window - to “catch” bootloader)
- no bootloader timeout (to return to application if no communication happens)

CANcrypt communication protocols supported
(see the book “Implementing Scalable CAN Security with CANcrypt” for details):

- Identify and Extended Identify
- Pairing (supporting the manufacturer key and the system integrator key)
- Generic Access (Read and Write)
  Offers authenticated CANopen SDO inspired access
- Optional: Key generation (including key save)
  Only manufacturer key can be used to erase keys and program new ones
- CANopen SDO inspired file transfer protocols (only used for transfer of code file)

3.2 Flash storage and parameters

Besides the application code, the following information is stored in the Flash/EEPROM area of the microcontroller:

- The code protection / manufacturer key (default 128-bit)
- The CANcrypt connection key (default 128-bit)
- Code flash status:
  - Flash is erased
  - Dirty (not empty, not completed)
  - Complete (flash programming completed, but not yet confirmed)
  - Confirmed (final confirmation from host received, authenticated and checksum verified)
- Optional configuration parameters for the commercial version:
  - Default CAN bit rate used
  - Default device / node ID
  - Default bootloader power-up delay (window to “catch” bootloader)
  - Bootloader timeout (stops if no messages received within timeout)
  - Optionally enable “match serial number” to only allow code protection files with matching serial number to be programmed
  - Optionally enable “reject firmware downgrades” to only allow upgrades as downgrades could allow installing an outdated version with known vulnerabilities

An application is only started if the code flash status is “confirmed” and a checksum check of the code flash completed successfully.

Once the code is programmed in Flash, the only method used to check the code’s integrity is a 32-bit CRC. The bootloader only starts application code after the 32-bit CRC has been verified, otherwise the bootloader remains active to wait for new code.

3.3 Configuring the bootloader and initial keys

The bootloader is delivered as a hex file without checksum and without encryption keys. These have to be added to the hex file before programming it into the chip. The tool BootloaderPostProcessor is used for this. Example:

```
>BootloaderPostProcessor.exe -s 0 -e FFCB -c FFCC -k FFD0 --
manufacturerkey=key_mf.txt --systemsintkey=key_sys.txt -i
cC_bload.hex -o cC_bload_sec.hex
```

This takes the bootloader hex file “cC_bload.hex” and adds the CRC-32 checksum, calculated over addresses 0h-FFCBh. The checksum is stored at FFCCh. The manufacturer and

system integrator keys are stored at address FFD0h and taken from file “key_mf.txt” or “key_sys.txt”, respectively. The generated hex file “cC_bload_sec.hex” is the one to program into the chip with Flash Magic. Use

```
>BootloaderPostProcessor -h
```

to learn about these and other options for this tool.

4 Preparing the application and code updates

4.1 Memory layout

Flash: The bootloader uses the first flash sector 0 from 0h-7FFFh, and the application is limited to sectors 1 and up with the last long word reserved for the CRC-32 checksum. This means the application flash range is 8000h-7’FFFBh.

EEPROM: The bootloader uses the second-last page at addresses 4010’BF00h-4010’BF7Fh. The application should never write to that page, otherwise the bootloader will only run with its default configuration. All other pages between 4010’8000h-4010’BEFFh may be freely used.

OTP: The bootloader reads word 2 from the OTP space, address 4001'5038h during startup and provides it as the 32-bit serial number in the extended identification entry [1018,4] as described in chapter 5.4.1 . If unprogrammed, the value reads 0.

4.2 Activating the bootloader

An application should have the ability to activate the bootloader. The trigger for this activation will be application-specific. As mentioned in chapter 1.2.2 , the mechanism is writing a long-word activation key into RAM followed by a reset. The address is 200’27FFCh and the value to write is 2165’4387h. Example:

```
*((unsigned int *)0x20027FFCUL) = 0x21654387UL;
...reset through watchdog or reset instruction.
```

4.3 File generation

Upon building a new firmware, the compiler system generates a hex file. A command line utility provided takes this file and further inputs to generate the protected code update file.

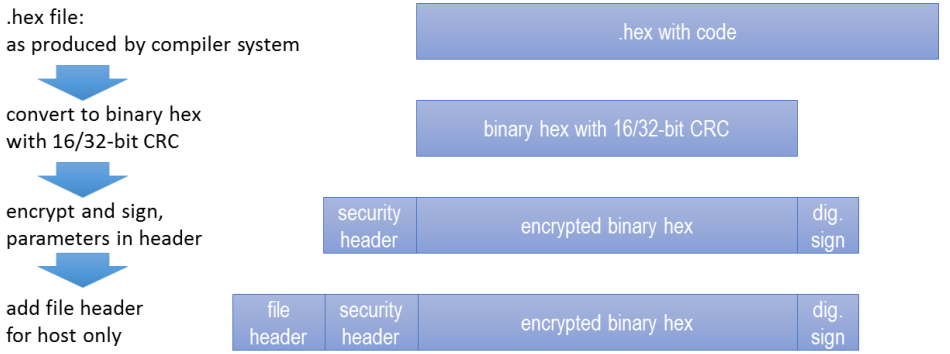


FIGURE – GENERATING PROTECTED CODE UPDATE FILES

The steps illustrated above are:

1. Code generated by compiler system and provided as ASCII hex file
2. Format converted from ascii to binary, to shrink file size and calculate embedded 32-bit CRC, add as own hex record. Using hexsum utility.
3. Generate a security header (see below for details), encrypt binary hex and generate a digital signature covering header and encrypted binary hex. Using Firmware Encrypter utility.
4. (Optional) add a file header for the host application to easily detect if a file is a protected code update file. The host-only file header is currently not used.

Example for step 2:

```
>hexsum.exe app.hex -b8000 -e7FFFB -c7FFFC -3 -l -i -x
```

This will generate the file “app_chk.bin”. Use

```
>hexsum.exe
```

to learn about the options for this tool.

Example for step 3:

```
>FirmwareEncrypter -e AES128GCM --fwmajor=1 --fwminor=0 -i
app_chk.bin -o app_sec.bin -k key.txt
```

This will generate the code update file “app_sec.bin” for a firmware version 1.0, using AES128-GCM encryption/authentication with the AES key from file “key.txt”. Use

```
>FirmwareEncrypter -h
```

to learn about these and other options for this tool.

The code update file (and only this one!) needs to be transferred to the system integrator / service technician performing the update.

4.4 File transfer and flash programming

The code update file is transferred to the bootloader in step 6 of the bootloader operation process (see 3.1 and 3.2). The bootloader receives the file (without file header, starts with security header) in segmented portions and works on segments as RAM is not big enough to store entire file.

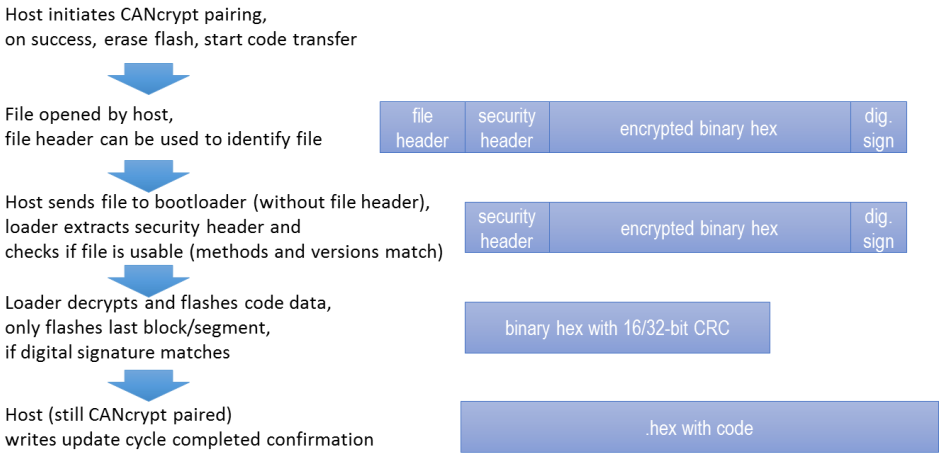


FIGURE – TRANSFERRING CODE UPDATE FILE TO BOOTLOADER

5 Implementation notes

5.1 Protected code update file format

```
   Begin file header, plaintext, no security, not send to bootloader
   | (currently unused)
   End of file header

   Begin of Init Vector entry, no security, plaintext
   | Length of IV in bytes (2 byte)
   | Init Vector (size as indicated)
   End of IV entry

 +Begin of authentication
 |
 | Begin security header (authenticated, not encrypted)
 | | File ID (4 bytes)
 | | File format version (1 byte)
 | | Min bootloader minor version (2 bytes, 0 for unknown)
 | | Min bootloader major version (2 bytes, 0 for unknown)
 | | App version minor  (2 bytes, 0 for unknown)
 | | App version major  (2 bytes, 0 for unknown)
 | | Chip serial number size (1 byte, 0 if not used)
 | | Chip serial number (size as indicated)
 | | Chip serial number reserved  (size as indicated)
 | | Encryption method used (1 byte)
 | | Hash method used (1 byte, don't care for AEG-GCM)
 | | Total input file size without padding (4 bytes)
 | End of Security header
 |
+|Begin of encryption
||
|| Encrypted input file with padding
||
|+End of authentication
+End of encryption for AES-GCM mode
|
|  Begin digital signature (SHA-256 hash or AES-GCM tag)
|  | Signature (size depends on method)
|  End of digital signature
|
+End of encryption for AES-CBC with SHA-256 mode
```

5.2 Bootloader activation

To activate the bootloader, an application has to write a 32-bit bootloader activation code to a reserved RAM cell and then execute a reset, for example using the watchdog. The microcontroller re-starts and the bootloader code is executed first. The bootloader checks the activation code. If it matches, the bootloader remains active.

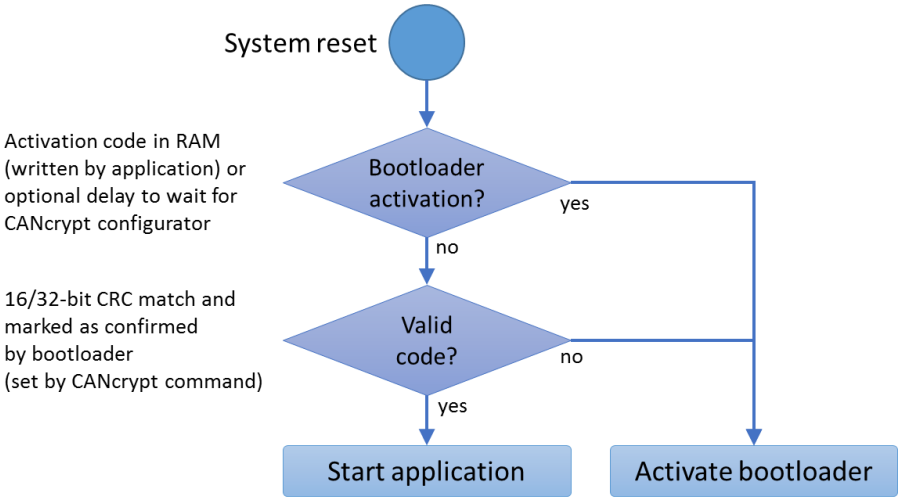


FIGURE – BOOTLOADER ACTIVATION

5.3 Bootloader state machine

Once the bootloader is activated, it first checks if there is a security delay to execute. The security delay increases with each failed CANcrypt pairing attempt, ensuring that brute-force-attacks (just trying different keys) take a long time to execute and therefore are impractical.

Once the delay is over, he bootloader waits for a CANcrypt configurator (e.g. the Flash Magic utility) to *initiate* the pairing sequence.

In paired mode, the bootloader accepts CANcrypt secure generic access commands (read and write of parameters supported), including activation of the code transfer mode.

On completion of the transfer and the programming of the code, the configurator still has to write the “update cycle completed” confirmation to the bootloader before the new application can be started.

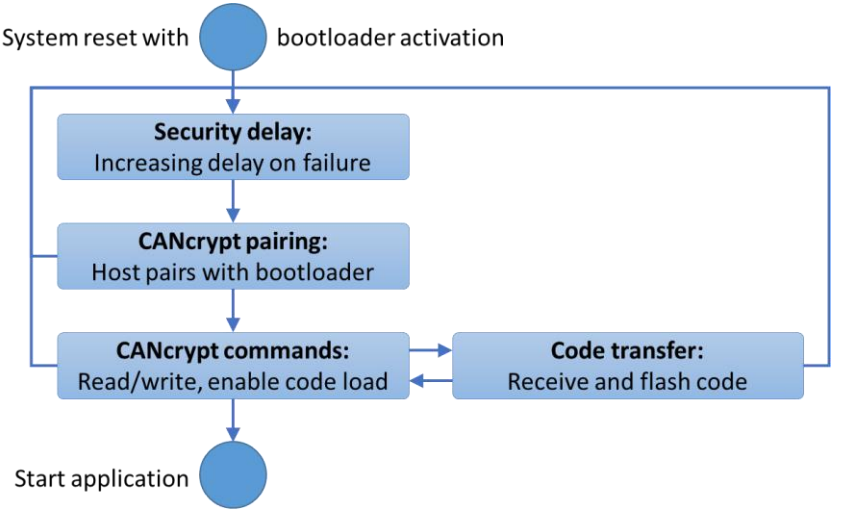


FIGURE – BOOTLOADER STATE MACHINE

5.4 Accessible parameters

5.4.1 Extended identification and status

These entries are all public and read-only. They can be read using the extended identify request.

Use	Index	Subindex	Type	Access
Vendor ID (0: no CANopen Vendor)	1018h	1	UNSIGNED32	RO
Product code (chip ID)	1018h	2	UNSIGNED32	RO
Revision num. (bootloader version)	1018h	3	UNSIGNED32	RO
Serial number (0 if not used)	1018h	4	UNSIGNED32	RO
CANcrypt version and support	5EF0h	1	UNSIGNED16	RO
CANcrypt address	5EF0h	2	UNSIGNED8	RO
CANcrypt status	5EF0h	3	UNSIGNED8	RO
Current key ID and length	5EF0h	4	UNSIGNED16	RO
System integrator public key ID	5EF2h	5	UNSIGNED32	RO
Manufacturer public key ID	5EF2h	6	UNSIGNED32	RO

PARAMETERS FOR IDENTIFICATION AND CANCRYPT STATUS

5.4.2 Secure access parameters

These entries can only be read or written when paired with a configurator.

Use	Index	Subindex	Type	Access
Program control	1F51h	1	UNSIGNED8	WO
Flash status	1F57h	1	UNSIGNED16	RW
Chip serial number	5100h	1-4	UNSIGNED32	RO

SECURE ACCESS PARAMETERS

6 Implemented CANcrypt protocols

[NOTE: This chapter contains selected chapters and sections from the book “Implementing scalable CAN security with CANcrypt”]

6.1 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.

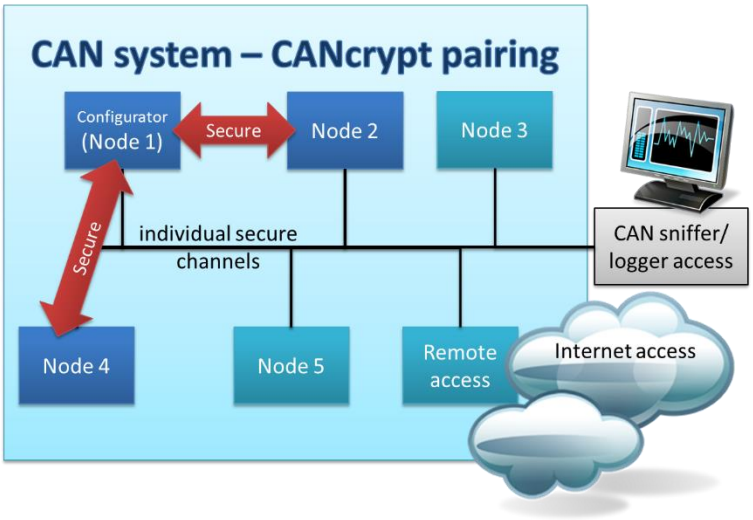
Security features are based on shared symmetric keys. There is a group key for all devices participating in the secure communication and a pairing key for secure channels between two devices. The secure pairing channel has a higher security level for use in system configuration or especially sensitive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

6.1.1 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.



SECURE CHANNELS IN A CAN SYSTEM

6.2 Basic functionality

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

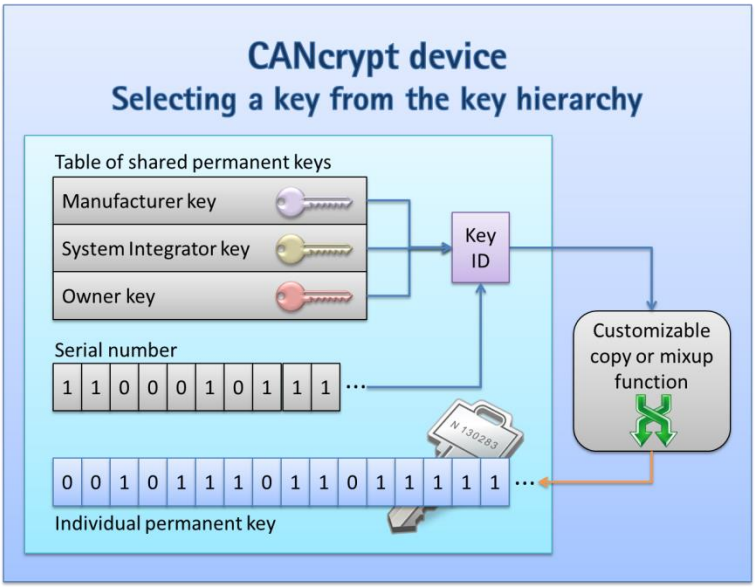
6.2.1 Key management and key hierarchy

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.



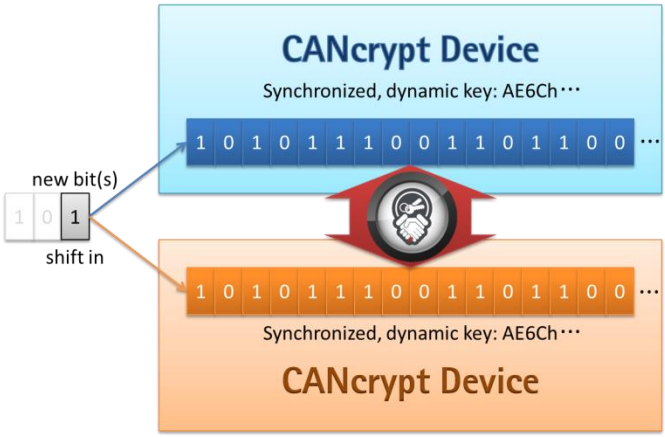
KEY SELECTION FROM KEY HIERARCHY

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.

6.2.2 Updating the shared dynamic keys

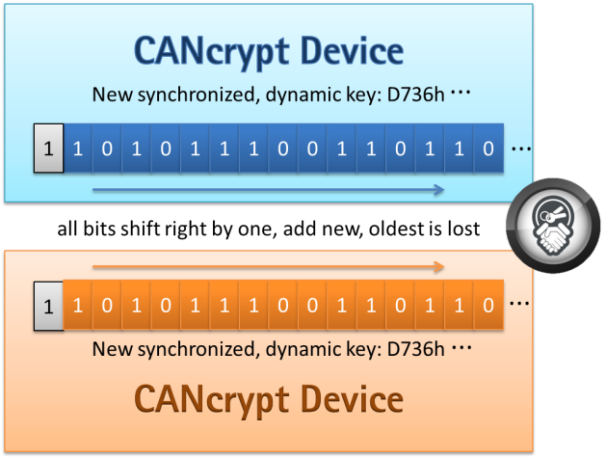
The dynamic key gets continuously updated. For a single pair of devices, a single new bit is generated randomly, initated by the configurator.

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.

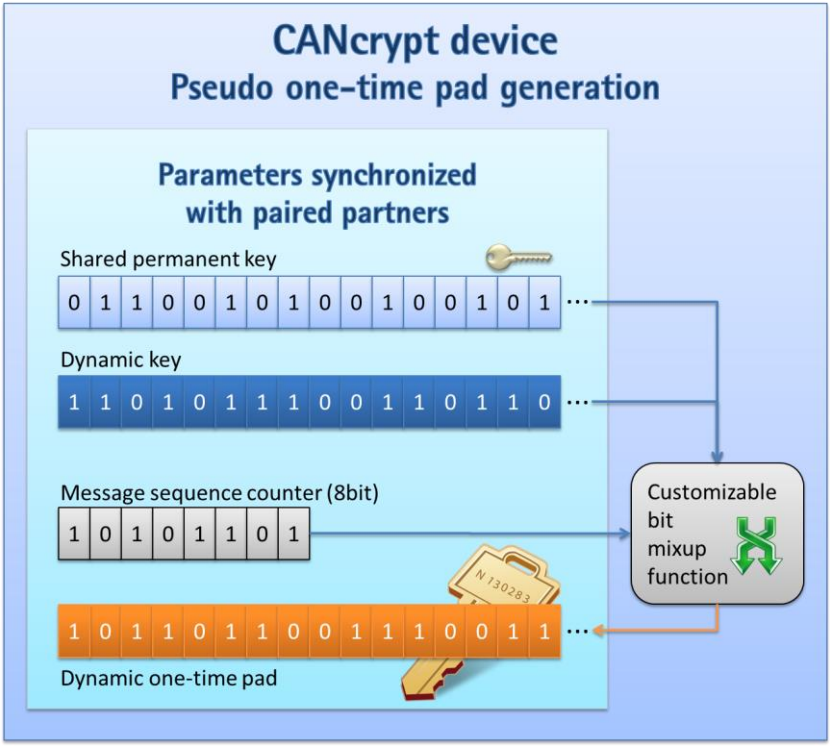


NEW BIT IS SHIFTED IN

6.2.3 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.

6.3 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.

6.3.1 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.

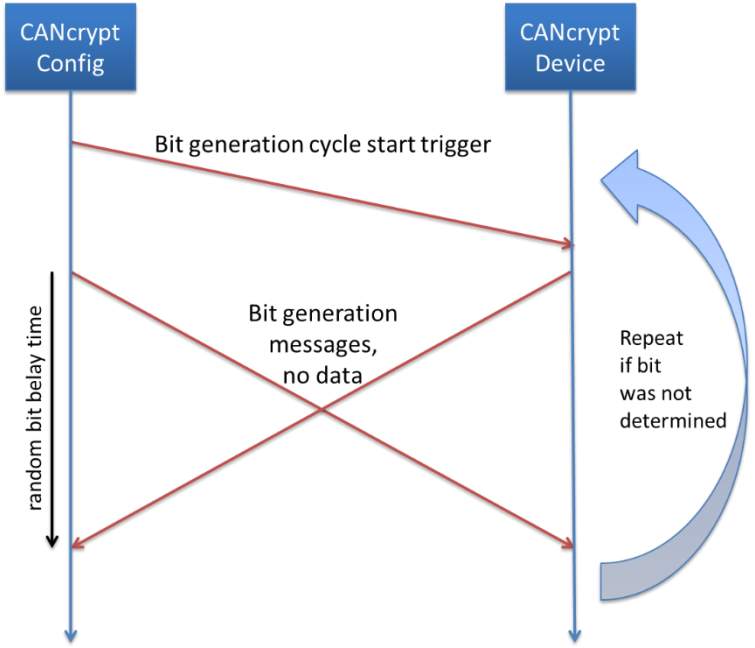
At the end of the bit select time window, a trace recording of the CAN messages exchanged will show one of the following scenarios:

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.

Collision avoiding variation

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
```

6.4 Common CANcrypt parameters

In this section, we describe the parameters required to maintain CANcrypt.

6.4.1 Device numbering and addressing

Address

In all CANcrypt request or command messages, a 4-bit value addresses the target CANcrypt device. A value of zero broadcasts to all devices (for example, used by the identify request). Values 1–14 are for CANcrypt devices 1–14. Address 15 is reserved for the CANcrypt configurator.

To simplify code optimizations, the addresses should be assigned incrementally starting with 1. In the CANcrypt implementation, a parameter can be set to the “highest address used”. If this is set to a value below 14, CANcrypt devices using an address higher than that value must not be used (besides the CANcrypt configurator).

6.4.2 The Keys

CANcrypt supports a number of permanent keys. This allows having multiple keys per device, such as a manufacturer key for bootloader access, a system key (created upon first startup of a CAN system), or further application-specific keys or session-limited keys. For any key stored in non-volatile memory, the size is in the range 128 –1024 bits.

The main keys used are the dynamic key and the permanent key. The permanent key is the non-volatile stored key used for the initialization of the current secure communication. The dynamic key is initialized from that permanent key (a direct copy or generated using a common mixup function) and continuously modified either based on the random bit-select cycles or via the bit-update request.

The last session key can store the dynamic key over a power cycle. If there is a proper shut down procedure before power down, the dynamic key can be saved as the last session key. On the next power up, the key is reloaded to the dynamic key, drastically shortening the initialization phase.

To globally identify the keys, CANcrypt uses 8-bit Key ID and Key length parameters. These values are used as described below.

Key ID

The Key ID is divided into a 3-bit major value and a 5-bit minor value.

The major value specifies one of eight key types and directly implements a key hierarchy. Higher values have a higher authority. The key erase command can be used only on keys that have the same or lower major value as the key currently in use.

The minor value plus specifies 32-bit segments within the key.

The key length value determines, if a key is used by itself without modifications or gets combined (mixed up) with the local serial number.

The values are mapped to UNSIGNED8 values. The major part uses the three most significant bits, and the minor part uses the five least significant bits.

Default use	Memory	Key ID major	Key ID minor	Length (bit)
Reserved		7		
Manufacturer key	NVOL	6	0–31	128–1024
System Integration key	NVOL	5	0–31	128–1024
Owner key	NVOL	4	0–31	128–1024
User key	NVOL	3	0–31	128–1024
Last group session key	NVOL	2	0–15	128–512
Dynamic pair session key	RAM	1	0–15	128–512
Dynamic group session key	RAM	0	0–15	128–512

THE KEY HIERARCHY

Key length

The Key Length is of type UNSIGNED8. To support a wide variety of key lengths with 8-bit encoding, the highest bit determines if the size is specified in bits or in other units as shown in the table below (Key Length Values Supported by CANcrypt).

Value	Interpretation
00h	Reserved
01h–20h	Key length in bits, 1–32
21h–7Fh	Reserved
80h	Single bit of dynamic key
81h–A0h	Key length in multiples of 32 bits, 1–32 (32–1024 bits)
A1h–C0h	As above, but key is combined with serial number
C1h–FFh	Custom, manufacturer specific sizes

KEY LENGTH VALUES SUPPORTED BY CANCRYPT

6.4.3 Status

This section describes the status information that must be provided by all participating CANcrypt communication partners.

Status

The CANcrypt status byte provides the following information and is the same for both the CANcrypt configurator and devices:

- Bits 0–1: Pairing status
  0: not paired
  1: pairing in progress
  2: paired
  3: pairing error
- Bits 2–3: Grouping status
  0: not grouped
  1: grouping in progress
  2: grouped, secure heartbeat enabled
  3: grouping error
- Bits 4–5: Result of last command or request
  0: unknown
  1: success
  2: ignored
  3: failure
- Bit 6: Reserved
- Bit 7: Key generation in progress
  When set, this device is participating in key generation

6.4.4 Controls

This section describes the control commands and requests available to the CANcrypt configurator and devices.

Request and commands

The 4-bit request value is used in most CANcrypt protocols.

Message	Type	Consumer Address	Request
Abort	event, response	1–15	0
Acknowledge	response	1–15	1
Alert	event	0	2
Identify	event	0	3
Pairing	request, response	1–15	4
Unpairing	request, response	1–15	5
Bit or key generation	request, response	1–15	7
Bit generation trigger	request	0	8
Generic data read	secure exchange	1–15	10
Generic data write	secure exchange	1–15	11
Extended Identify	request, response	1-15	13

REQUESTS USED BY CANCRYPT DEVICES AND CONFIGURATOR

The requests and commands in the table “Requests used by CANcrypt devices and configurator” are used by both devices and the configurator in the same manner.

There is one exception: the identify request and response. When used by the configurator, these requests have extra parameters.

6.4.5 Methods

CANcrypt supports a variety of algorithms and features. The parameters selecting these are listed below.

Method

The 4-bit method parameter selects the base algorithm used to generate the random bit and specifies a security method.

- Bits 0–1: Security functionality
  0: Basic security
  1: Regular security
  2: Advanced Security
  3: Custom security
- Bit 2: Bit generation method, set to 1 for random delay, otherwise direct, immediate reply to trigger message.
- Bit 3: Number of bit generation messages used. When set, 16 bit generation messages are used, else 2.

The security settings influence the bit-generation cycle, authentication, and encryption.

Bit generation:
After each bit-generation cycle, the customizable function UpdateBit() is called and can flip the bit generated, for example depending on the permanent key. This increases security for cases where an intruder has physical access to the CAN system as the intruder cannot easily determine when a new bit generated is 0 or 1. In addition, bit stuffing is used.. This ensures that the Mixup() function used for authentication and encryption does not use a value with all the same bits.

Authentication:
The signature used for messages is 16 bits. The signature is generated by the combination of a checksum that is encrypted using a bit mixup of the current dynamic key and the message counter. In basic mode, the checksum is calculated in Fletcher style with the initialization generated from the permanent or dynamic key. In regular mode or higher a 16bit CRC checksum is used. In advanced mode AES-128 is used for encryption/decryption.

Encryption:
The encryption is based on a mixup of the current dynamic key.

6.4.6 Functionality

Individual CANcrypt functionality may be enabled or disabled.

Functionality

If a corresponding bit is set, the functionality is enabled

- Bit 0: authentication used
- Bit 1: encryption used
- Bits 2–3: reserved

6.4.7 Timings

CANcrypt uses various timings and timeouts. To minimize the number of definitions, specific values are defined as a group.

Timeout

The 4-bit timeout value defines the timing and timeout options CANcrypt uses:

- Bits 0–1: timing used
  0: fast
  1: medium
  2: slow
  3: custom timing
- Bits 2–3: reserved

Values 0–2 activate the defined timings in the table below, Timeouts Used by CANCrypt). Value 3 selects custom, manufacturer-specific timings.

CANcyrpt message timeout:
If a CANcrypt message contains a request, requiring a response, then the transmitter uses this timeout to wait for an response from the device addressed. If no response is received within this time, the transmitter internally marks the addressed device as not present.

Secure message timeout:
Every secure message combination using a preamble and one or multiple following data messages have to transmit the messages back to back on the network. On the receiving side the data message is only considered to be received in time, if the time since reception of the preamble does not exceed this timeout.

Timeouts	Fast	Medium	Slow
CANcrypt message timeout (request to response)	100 ms	200 ms	400 ms
Bit select cycle time for random delay method	25 ms	50 ms	100 ms
Bit select cycle time for direct response method with no delay	10 ms	25 ms	50 ms
Bit select cycle random delay window	0–16 ms	0–32 ms	0–64 ms

DEFAULT TIMEOUTS USED BY CANCRYPT

Bit select cycle time and delay window:
The key- or bit-generation cycle time is a fixed value, the CANcrypt system tries to determine one bit per cycle. If the method with delays is used (each participant transmits their claim message randomly within a time window), then the maximum value for this delay is defined.

7.1 Basic protocol elements

The basic protocol elements include selecting CAN message identifiers, protocols for events like alerts or aborts, and the random-bit-generation cycle, which is used by multiple CANcrypt protocols.

7.1.1 CAN message identifiers

We strongly recommend that the CAN message identifiers be hard coded so they cannot be reconfigured through CAN communication during operation. Otherwise, attackers could try to reconfigure the CAN ID usage. On success, they would be able to logically disconnect one of the secure devices. Doing so would be a first step in an attempt to replace a secure device with a device provided by the attacker.

For the CANcrypt configurator and devices, we need one CAN message identifier for the device's main CANcrypt command, response and status message.

To simplify implementation, the CANcrypt devices should use up to 15 consecutive identifiers. The CANcrypt configurator uses the first identifier. The identifiers should be high priority (low value). When used as preamble to a high priority message, a low-priority identifier might cause delays.

The default CANcrypt message IDs are 171h to 17Fh. The CAN message IDs 172h –17Fh are used by the CANcrypt devices, and the CANcrypt configurator uses 171h. All devices must receive all CANcrypt message IDs and respond to requests or commands received.

The bit-generation cycle requires two CAN message IDs. Key generation or key update is a background process and may be of lower priority.

The default CANcrypt bit-generation message IDs are 6FEh and 6FFh. These are used by both the CANcrypt configurator and devices in the bit-generation cycle for generating keys or for pairing. The configuration must ensure that at any time, only one bit-generation cycle is active.

The default CAN message IDs are values that are reserved, and thus otherwise unused, by CANopen. Depending on the protocol or application, other identifiers may be used.

CAN ID	CANcrypt use
172h–17Fh	CANcrypt message of CANcrypt devices 2–15
171h	CANcrypt message of CANcrypt configurator
6F0h–6FFh	Bit-selection messages for random-bit-select cycle
6E1h–6EEh	Optional debug messages from CANcrypt devices

DEFAULT CAN IDS USED BY CANCRYPT

7.1.2 CANcrypt message common contents

The first two bytes of the CANcrypt message are identical for all requests and responses. They contain:

address (4 bits): destination device
access (4 bits): message identification request, response or event
status (8 bits): current status byte

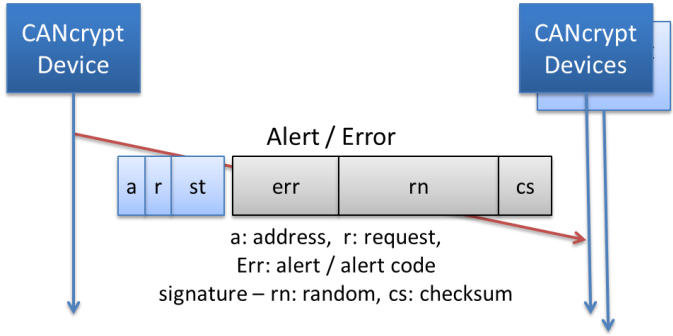
The address is the CANcrypt device number for the message's destination. Set to 0 for a broadcast to all devices or 1 for the configurator.

The access information identifies which request, response, or event the message contains. See section 6.4.4 for a complete list of all values and section 6.4.3 for a description of the current status byte.

The access type determines how many additional bytes follow these first two bytes and what information the bytes contain.

7.1.3 Alerts and errors

At any time, any CANcrypt device may generate an alert to signal that an error or intruder detection occurred. By itself, these signals are not secure.



ALERT OR ERROR SIGNAL

Parameters used (8-bit unless noted otherwise):

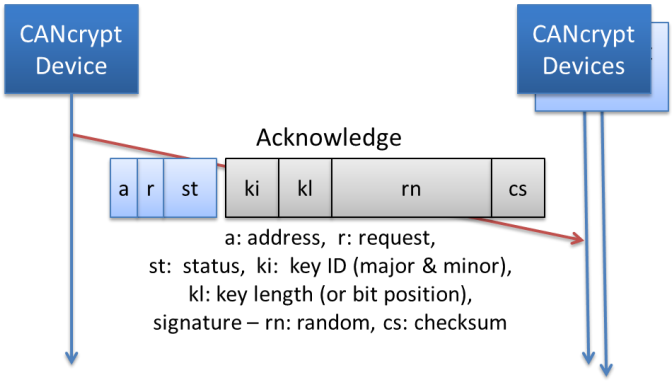
address (4 bits):	0, broadcast
request (4 bits):	2, alert
status:	current status byte
error (16 bits):	error code,
	high byte for error / alert code
	low byte for manufacturer specific information
sign. (32 bit)	signature, 3 random bytes, 1 byte checksum,
	all encrypted based on dynamic key

Code	Interpretation
80h to 8Fh	Intruder alert
90h to 9Fh	Key generation error or timeout
A0h to AFh	Pairing error or timeout
D0h to DFh	Generic access error or timeout
F0h to FFh	Manufacturer specific

CANCRYPT ERROR AND ALERT CODES

7.1.4 Acknowledge or Abort

All protocols consisting of a sequence of messages may use the messages acknowledge and abort. Aborts may be used at any time by any of the involved communication partners to abort (end) the sequence. Acknowledge may only be used as specified by the individual sequence. By itself, these signals are not secure.



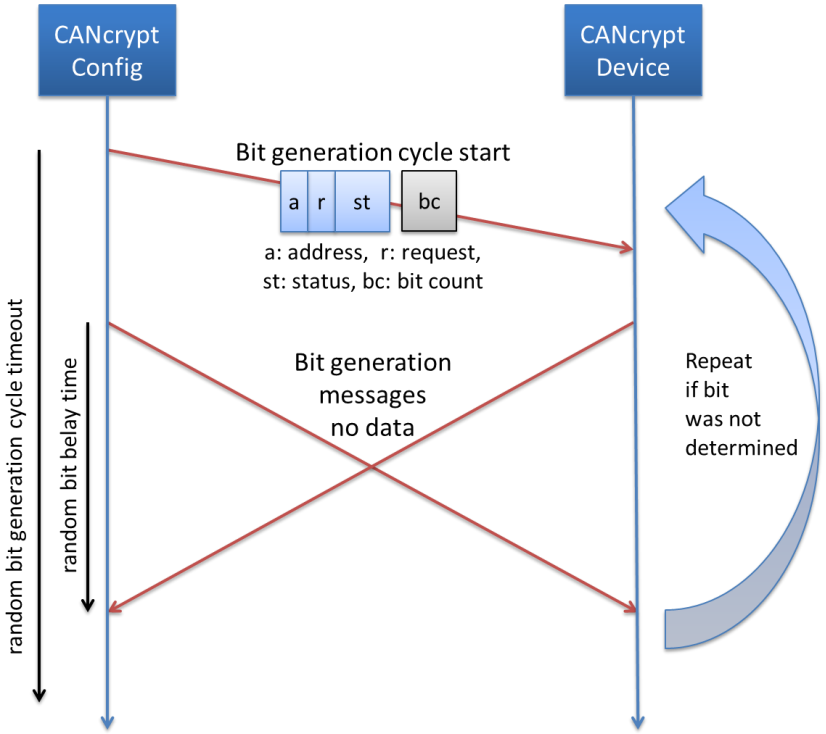
ACKNOWLEDGE OR ABORT

Parameters used (8-bit unless noted otherwise):

address (4 bits):	1–15, device address
request (4 bits):	0, abort or 1, acknowledge
status:	current status byte
key ID:	if key is involved, ID of key
key length:	if key is involved, length of key
sign. (32 bit)	signature, 3 random bytes, 1 byte checksum, all encrypted based on dynamic key

7.1.5 Sub-protocol for bit-generation

Several protocols require the generation of one or multiple shared bits. Each bit-generation uses the cycle outlined in this section.



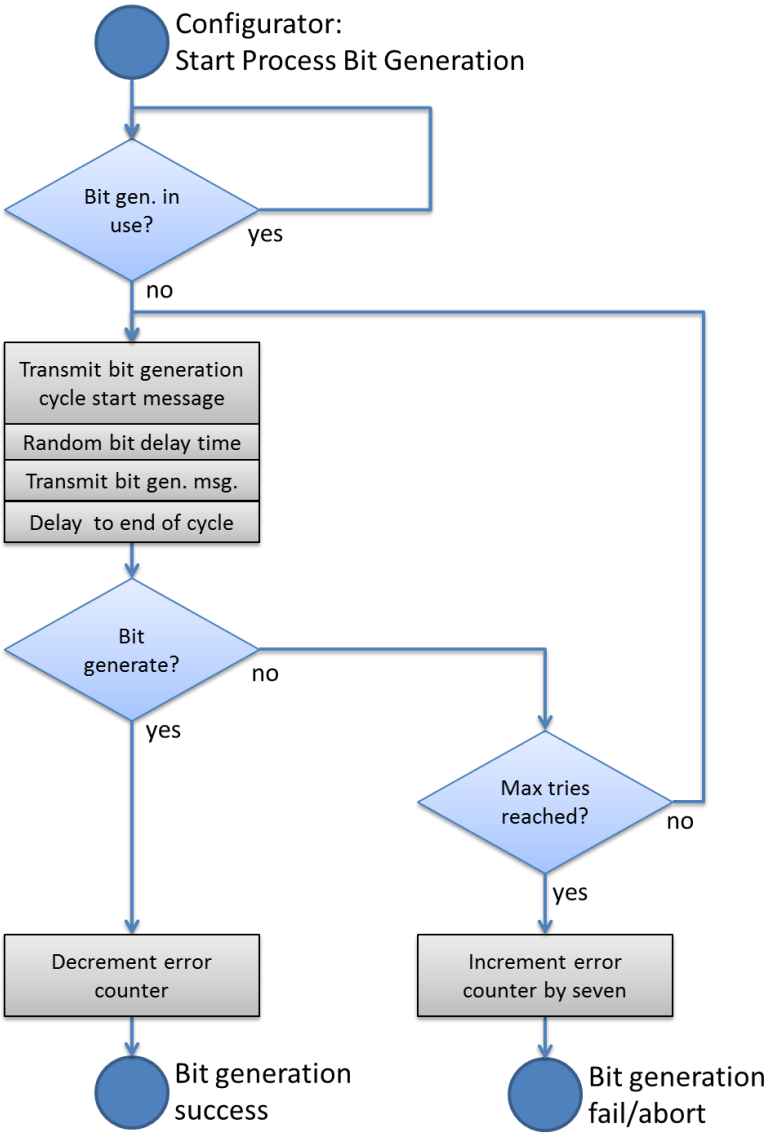
THE BIT-GENERATION CYCLE PROTOCOL

Parameters used (8-bit unless noted otherwise):

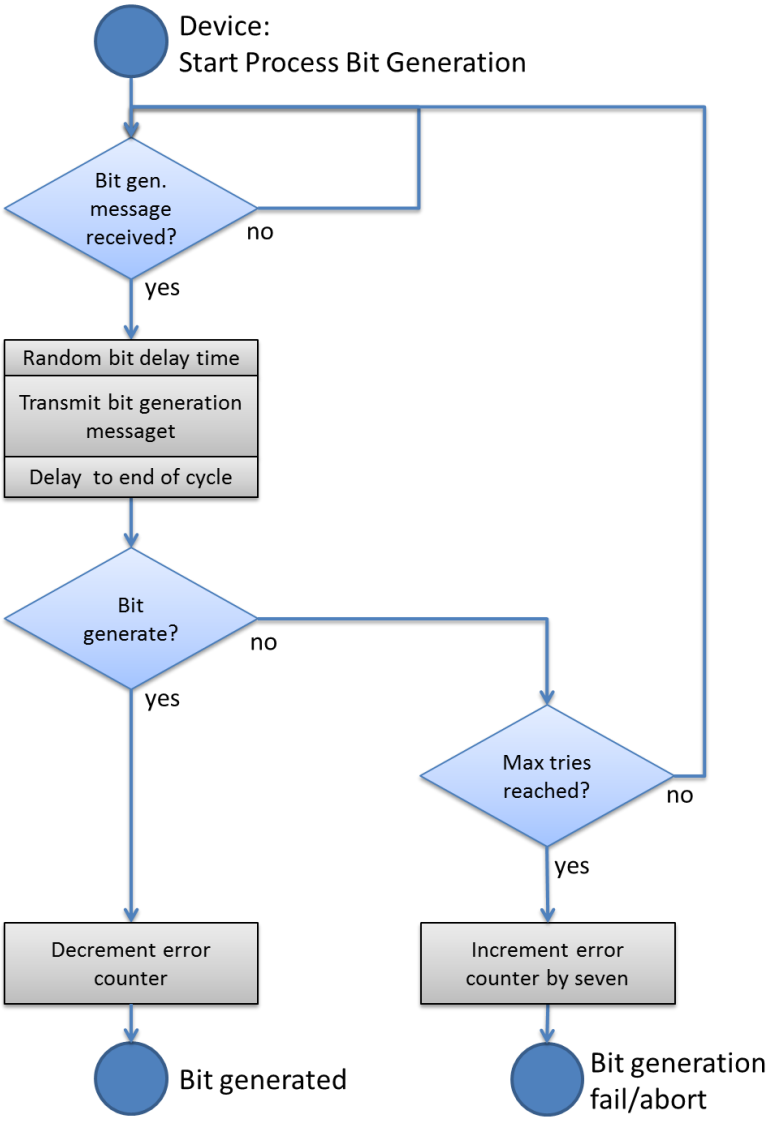
address (4 bits):	1–15, device address
request (4 bits):	4, pairing
	7, bit or key generation
status:	current status byte
bit count:	counting down from 31 to 1 or
	80h if this is a single bit for the dynamic key

Each cycle starts with a trigger message from the CANcrypt device initiating the bit-generation. Depending on the mode used, each device transmits its chosen bit-select message immediately or with a random delay. If a bit was not determined (both partners used an identical message), the cycle is repeated

The following flow charts illustrate the processes executed internally in the CANcrypt Configurator and CANcrypt device during the bit-generation cycle.



THE BIT-GENERATION CYCLE - CONFIGURATOR



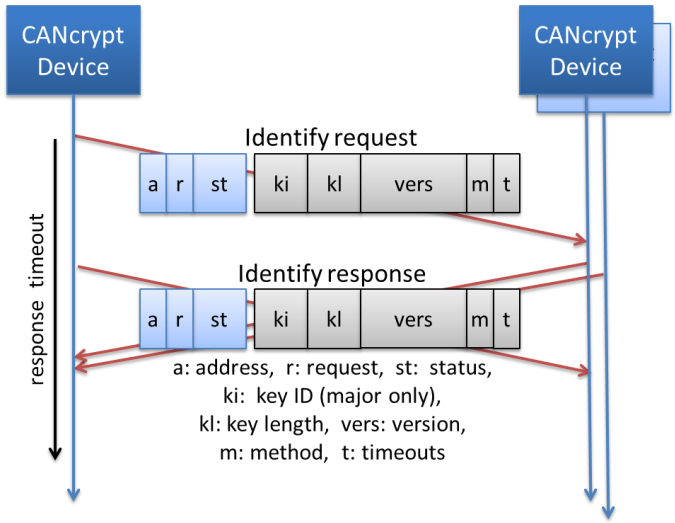
THE BIT-GENERATION CYCLE - DEVICE

7.2 Unpaired communication

The identification and extended identification requests may be sent from “any” CANcrypt communication partner. They allow reading basic identification information including the public key ID.

7.2.1 Identification

The identification message verifies if the CANcrypt devices on the CAN system are compatible with each other.



THE IDENTIFICATION REQUEST PROTOCOL

Parameters used for request (8-bit unless noted otherwise):

address (4 bits):	0, broadcast
request (4 bits):	3, identify
status:	current status byte
key ID:	key ID of the key that the manager requests to use
key length:	length of key
version (16 bits):	CANcrypt version number
method(4 bits):	method requested or supported
timeout(4 bits):	timeout requested or supported

The Identify request may come from any CANcrypt device and includes information about the key that should now be used and the version, method, and timeout. The chan-

nel number is set to the connection number of the device addressed (1–15). If set to zero, the identify request is a broadcast to all devices, and all devices must reply.

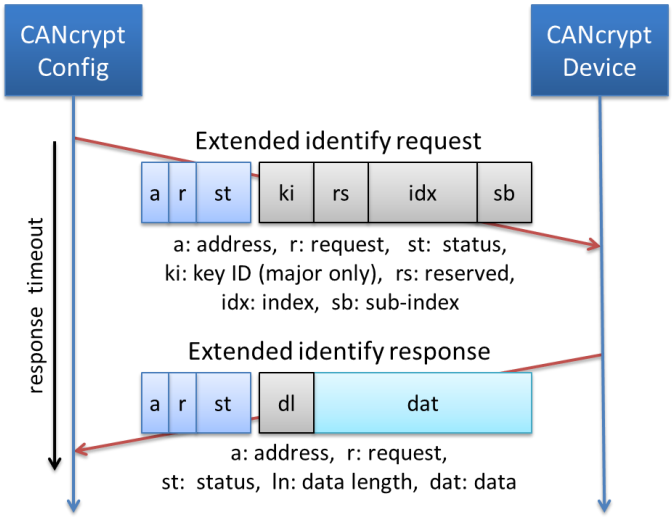
Parameters used for response (8-bit unless noted otherwise):

address (4 bits):	0, broadcast
request (4 bits):	identify
key ID:	confirms the requested key ID
	if not available, alternate key
key length:	length of key
version (16 bits):	CANcrypt version number
method (4 bits):	confirms requested method
	if not supported, alternate method
timeout (4 bits):	confirms requested timeout
	if not supported, alternate timeout

The CANcrypt device sends its response setting address to zero and copies its own version number into it. The key information, method, and timeout data is copied only if the device supports these parameters. A device that does not support a feature must modify the parameter in the response to indicate what the device can support. The requestor then has the option to send another identify request with different parameters, for example requesting to use a different key.

7.2.2 Extended Identification

The CANcrypt configurator has the option to request more detailed information using the identify request. To address the data, the index and sub-index system of CANopen is used. Devices not implementing CANopen shall at least implement the identity object as defined in the table below.



THE EXTENDED IDENTIFICATION PROTOCOL

Parameters used for extended request (8-bit unless noted otherwise):

address (4 bits):	1–15, device address
request (4 bits):	13, identify
status:	current status byte
key ID:	key ID requested
reserved:	0
index (16 bits):	index to information requested
sub-index:	sub-index to information requested

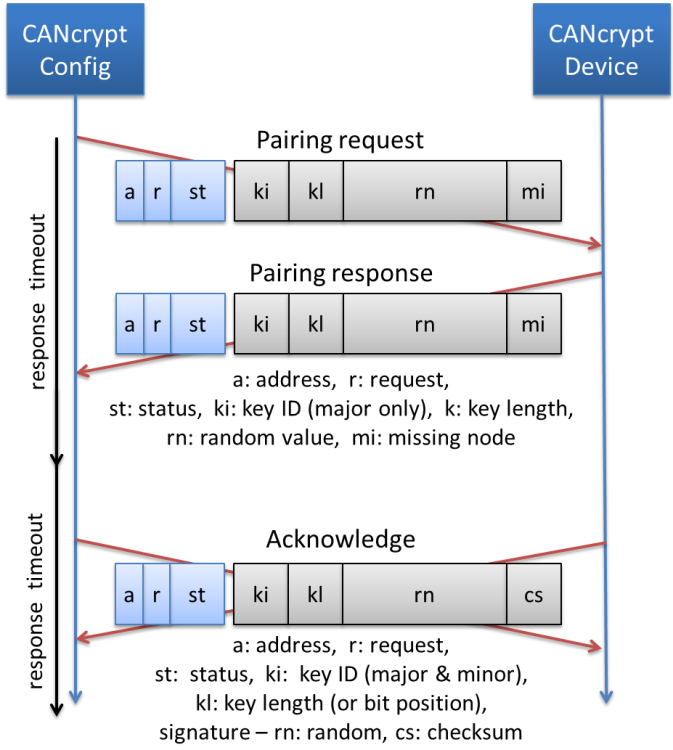
Parameters used for extended request (8-bit unless noted otherwise):

address (4 bits):	1, to address the configurator
request (4 bits):	13, identify
status:	current status byte
length:	length of data in bytes (1–4)
data:	data field

7.3 Pairing

In addition to the grouping method, CANcrypt supports the pairing of two devices. Paired devices have an individual security channel. Pairing is intended for communication between the configurator and a device.

7.3.1 Pairing with a single device, open a channel



THE PAIRING PROTOCOL

The pairing protocol establishes a secure point-to-point communication based on the selected key. The process includes generating a pair dynamic key from the selected key.

Parameters used for request (8-bit unless noted otherwise):

address (4 bits):	1–15, device address to pair with
request (4 bits):	4, pairing
status:	current status byte, pairing in process
key ID:	key ID of key to use as a start
key length:	length of this key
rand (24 bits):	random value used for key initialization, this value shall not change during the startup cycle
node missing:	1

Parameters used for response (8-bit unless noted otherwise):

address (4 bits):	1–15, address of requesting device
request (4 bits):	4, pairing
status:	current status byte, pairing in process
key ID:	confirming key ID
key length:	confirming length
rand (24 bits):	random value used for key initialization, this value shall not change during the startup cycle
node missing:	0

The initiator starts with the pairing request message. Unless the key selected is the last session key, the address value must be in the range 1–15 to select one specific pairing partner.

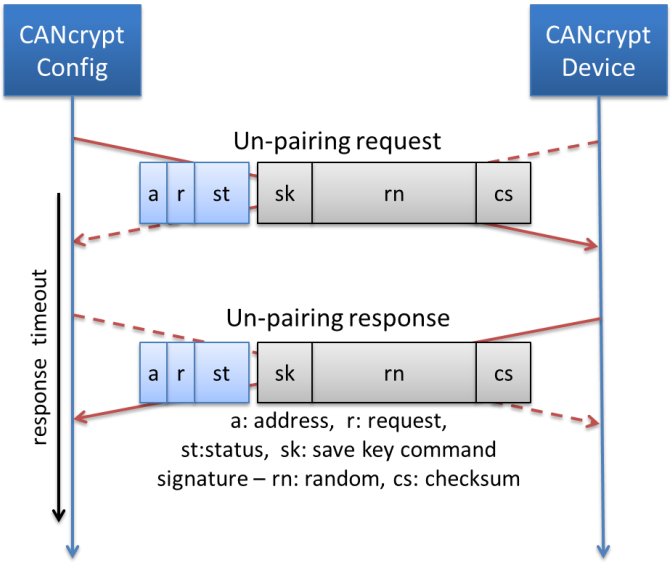
The CANcrypt device addressed confirms the request by returning the request if the device has the key (ID and length) available. Otherwise the device replies with the next higher key ID and length available in the device, the protocol aborts, and the manager needs to start over).

The initiator now starts the subprotocol random-bit-generation to generate as many bits as needed to address any bit in the specified key. If the key length is 256 bits, eight bits are needed. If the key is 1024 bits, 10 bits are needed.

The dynamic key is initialized by copying bits from the selected key and applying a mixup function using the random value generated. This method ensures that the dynamic key is not always initialized with the same value.

7.3.2 Unpairing

Any paired device may request to close the secure connection at any time.



THE CLOSE SECURE CHANNEL PROTOCOL

Parameters used for request and response (8-bit unless noted otherwise):

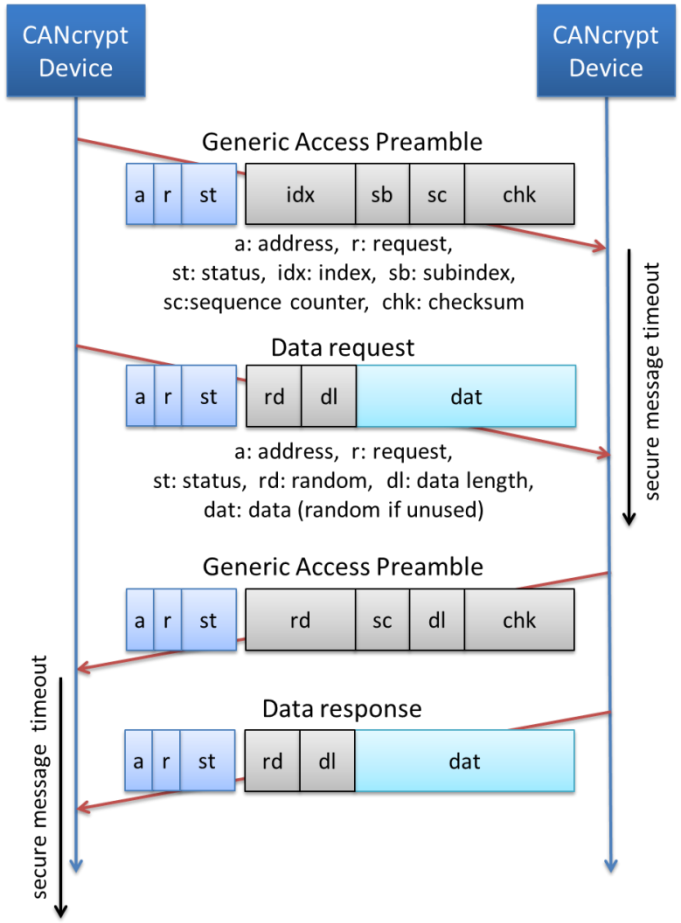
address (4 bits):	1–15, device address
request (4 bits):	5, unpair
status:	current status byte
sign. (32 bit)	signature, 3 random bytes, 1 byte checksum, all encrypted based on dynamic key

7.4 Paired communication

Once configurator and device are paired, generic data object access is used.

7.4.1 Secure generic data object access

Secure generic data object access is a transfer mode that allows paired CANcrypt devices to directly exchange data using CANcrypt messages. A 16-bit index and 8-bit sub-index address the data within the devices. The mode is compatible with CANopen and can also be used generically to address data available in the devices.



THE GENERIC ACCESS PROTOCOL

This transfer mode is also used by the configurator to erase, set, or save permanent keys.

Parameters used for generic access preamble request and response (8-bit unless noted otherwise):

address (4 bits):	1–15, address of paired partner
request (4 bits):	10, generic read access
	11, generic write access
status:	current status byte
index (16 bits):	index addressing the data in device
sub-index:	sub-index addressing the data in the device
seq. counter:	device's secure transmit counter
check (16 bits):	checksum of preamble and data

Parameters used for data request (8-bit unless noted otherwise):

address (4 bits):	1–15, address of paired partner
request (4 bits):	12, generic access data
random:	reserved, random value
data length:	length of data in bytes
data (32 bits):	on read: fill with random data,
	on write: data to write, fill unused bytes
	with random data

Parameters used for data response (8-bit unless noted otherwise):

address (4 bits):	1–15, address of paired partner
request (4 bits):	12, generic access data
random:	reserved, random value
data length:	length of data in bytes
data (32 bits):	on read: data requested, fill unused bytes
	with random data
	on write: fill with random data