CAN-FD stack porting and secure bootloaders

presented by

Olaf Pfeiffer
Embedded Systems Academy

www.esacademy.com
Webinar Contents

- Review of CAN-FD basics (from part 1)
- Implications for Higher-Layer Protocols
  - CANopen, J1939 and others
- Implications for Bootloading
- Security risks, ransomware
- ESAcademy’s Secure Bootloader
  - Protection levels
  - CANcrypt basics
  - Key management
  - Bootloader operation
  - LPC546xx implementation
Embedded Systems Academy

- Founded 1999
- Services
  - Consulting
  - Training
- Firmware
  - CANopen stack
  - J1939 stack
  - Bootloader
- Software
  - NXP’s Flash Magic
  - CANopen Magic
  - CANcrypt
- Participate in CANopen standardization

Blog: [www.esacademy.com/blog](http://www.esacademy.com/blog)

[www.flashmagictool.com](http://www.flashmagictool.com)
[www.canopenmagic.com](http://www.canopenmagic.com)
[www.cancrypt.eu](http://www.cancrypt.eu)
Review from part I of this webinar
FD: Flexible Data(rate)

CAN-FD BASICS
Differences between CAN and CAN-FD

- **Mixed bitrates**
  - “Nominal rate” for control (arbitration, control, ACK)
  - “Data rate” (multiple of nominal) for data field and CRC
    - Limited by transceivers in practice
    - Need FD-compliant transceivers above 1Mbps

- **More data per frame**
  - Up to 64 bytes instead of 8
  - Allows for higher throughput

- **Bus topology and wiring stays the same**
  - Same networking costs
  - More sensitive on higher rates

- **NOTE: if CAN-FD is enabled,**
  **ALL devices connected must support CAN-FD**
  - Exception: CAN-FD “ignoring” transceivers
CAN-FD Message Frames

Diagrams © CiA
Higher Layer Protocols like CANopen, J1939, others

PROTOCOL IMPLICATIONS
It’s a hard transition

- CAN and CAN-FD can not easily be mixed
  - A classic CAN controller not capable of CAN-FD will destroy CAN-FD messages with error frames

- If CAN-FD is enabled, all participants must support it

- Therefore higher layer protocols do not necessarily have to be backward compatible
  - There is no „mixed“ operation, just either / or

- First step is to re-pack pre-defined data messages
  - Now up to 64 bytes (instead of 8) available

- This is work in progress...
CANopen-FD is still under development
First demonstrators have been shown
Support of 64byte message length for „PDO“
  - Process data objects can now contain more data
  - As a result less CAN-IDs are required per node
New transfer mode „USDO“ instead of „SDO“
  - Universal Service Data Object
    - Request / Response communication
    - Fully meshed (every device can do this)
    - Any size (segmentation included)
    - Broadcast
CAN-in-Automation (CiA) members have mapped SAE’s J1939 application profile to the CAN FD data link layer

- Corresponding CiA 602-2 specification to be released
How to upgrade existing CAN code

- If a 3rd party communication stack is used, upgrading to CAN-FD should be done by developers of stack.
- If properly done, should be possible to do with minimal changes to application interface.

ESAcademy‘s Micro CANopen Example:

- All parameters and all data communicated is in an object dictionary (kind of look up table).
- API addresses Objects in this dictionary, then application does not need to make any modifications.
  - Unless complete new features are used
  - Example: mass broadcast
Code updates via CAN or CAN-FD

BOOTLOADER IMPLICATIONS
It’s about code size and update time

- **Speeding up code updates was one of the driving factors behind the development of CAN-FD**
  - Tendency is that code gets bigger
  - 128k update on classical CAN can take minutes
    - Main issue is segmentation not speed
      - For reliable transfer only segment by segment
      - Request-Response-Request-Response...
    - Over the thumb estimate: one segment per 3-5ms

- **Data transfer per segment**
  - One byte per segment used for flow control
  - Data bytes per segment
    - Classical CAN: 7 bytes
    - CAN-FD: 63 bytes

- **Conservative expectation is that code updates are executed 8 times faster**
  - 128k update on CAN-FD within 5 seconds
Compatibility issues

- When CAN-FD is actively used
  - All connected and powered up CAN controllers must support CAN-FD
    - Else error frames will be generated by classical CAN devices

- An application uses classical CAN, can CAN-FD be used for code updates only?
  - Possible if during the bootloading process all non CAN-FD capable devices are disconnected or powered down
Code update security issues, ransomware

BOOTLOADER RISKS
What could possibly go wrong?

If code falls into the “wrong hands”, …

- … could it be easily copied to other devices?
  - Programmed into a copy of the original hardware?

- … could intellectual property be extracted?
  - Re-engineering of code and used elsewhere?

- … could an attacker modify it?
  - Before it gets programmed into your device, introducing malicious code?
  - Could Embedded Ransomware lock the device?
Attacker access options to CAN or CAN-FD

- Unlimited physical access
- CAN sniffer/logger access
- Internet access

**Attack vectors**

CAN system

- Node 1
- Node 2
- Node 3
- Node 4
- Node 5
- Remote access
ESAcademy’s CAN(-FD)

SECURE BOOTLOADER
CANcrypt Basics

- Security framework supporting various methods
- Secret key generation and exchange
- Pairing and grouping
- Encrypted and authenticated communication
- Minimal authentication using a secure heartbeat
CANcrypt: Secret bit generation

- Cycle initiated by configurator
- In random time window both transmit randomly X or Y
- If window contains XX or YY, start over
- If window contains XY or YX, generated bit is 1, if configurator sent X, else bit is 0
- 256bit key in 4-6s
CANcrypt: Pairing and key exchange

- Secure connection between two devices
  - based on symmetric key
- Initiated by configurator
- Uses CANcrypt bit generation cycle
- Intended use
  - key generation and exchange
  - device setup or configuration
  - crucial commands like bootloader activation
CANcrypt: Key management

- Key hierarchy
  - symmetric

- Different keys can have different authorities

- Bootloader access limited to manufacturer and system integrator

- Optional: combined with serial number

CanCrypt device
Selecting a key from the key hierarchy

Table of shared permanent keys

- Manufacturer key
- System Integrator key
- Owner key

Key ID

Serial number

1 1 0 0 0 1 0 1 1 1

Individual permanent key

0 0 1 0 1 1 1 0 1 1 0 1 1 1 1 1
CANcrypt: Key identification

- How can a key management system remember which key was installed where?
- Each key is associated with a unique 32-bit key ID assigned and stored when installing the key

Table of shared permanent keys

| Manufacturer key | System Integrator key | Owner key |

Public, readable

- Manufacturer key ID
- System Integrator key ID
- Owner key ID

- The key ID can be read at any time (public info)
- Service case: service utility reads public ID and then checks if it has a matching key in its database
Security levels supported

- **Global and Local protection**
  - Global (code, manufacturer): code send via Internet
  - Local (connection, system integrator): bootloader activated locally

---

**Application code generation (encryption utility)**
- Code protection key
- Encrypted application code

**CANcrypt Configurator (configuration utility)**
- CANcrypt connection key
- Encrypted application code

**CAN FD**
- Application code

---

**LPC546xxx Flash Memory**
- Secure Bootloader
- Code protection key
- CANcrypt connection key
- Optional application keys
- Application flash status

---

**PC at manufacturer generating new FW**

**PC or diagnostic tool at system integrator**

**Embedded system receiving FW update**
Flashbootloader and initial key(s)

- Must happen in a trustworthy environment
  - No difference to public/private key systems, private key must be protected

- Key generation and installation is where any “root of trust“ begins

- Preferably in between
  - Production
  - Delivery

- Here: supported by FlashMagic utility
Secure bootloader activation

Application writes "BOOT" code to RAM and resets

System reset

Activation code in RAM (written by application) or optional delay to wait for CANcrypt configurator

Bootloader activation?

Valid code?

Start application

Activate bootloader
Bootloader state machine (once it is activated)

- System reset with bootloader activation
- Security delay: Increasing delay on failure
- CANcrypt pairing: Host pairs with bootloader
- CANcrypt commands: Read/write, enable code load
- Code transfer: Receive and flash code
- Start application
Generating the code update file (utility provided)

- **.hex file:**
  - As produced by compiler system
  - Convert to binary hex with 32-bit CRC
  - Encrypt and sign, parameters in header
  - Add file header for host only

<table>
<thead>
<tr>
<th>.hex file</th>
<th>.hex with code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Binary hex</td>
<td>Binary hex with 32-bit CRC</td>
</tr>
<tr>
<td>Security header</td>
<td>Encrypted binary hex</td>
</tr>
<tr>
<td>Dig. sign</td>
<td>Dig. sign</td>
</tr>
<tr>
<td>File header</td>
<td>Security header</td>
</tr>
<tr>
<td>Encrypted binary hex</td>
<td>Dig. sign</td>
</tr>
</tbody>
</table>

---

CAN-FD II
June 2017

Slide 28
Contents of the security header

- **Bootloader version number required**
  - Ensure that bootloader matches to file

- **Firmware version number**
  - Only allow upgrades not downgrades

- **Serial number of destination chip**
  - If set, only allow to be programmed in matching device

- **Encryption method**

- **Encryption parameters**
  - key info, vectors, size

- **Signature method**

- **Signature parameters**
  - key info, vectors
Code update file processing

Host initiates CANcrypt pairing, on success, erase flash, start code transfer

File opened by host, file header can be used to identify file

Host sends file to bootloader (without file header), loader extracts security header and checks if file is usable (methods and versions match)

Loader decrypts and flashes code data, only flashes last block/segment, if digital signature matches

Host (still CANcrypt paired) writes update cycle completed confirmation
ESAcademy’s CAN(-FD) secure bootloader
LPC54618 IMPLEMENTATION
LPCXpresso54618 CAN-FD Kit

Development platform for LPC546xx Series

- LPC54618 MCU running at 180MHz
- 128Mb Micron SDRAM
- 128Mb Micron quad SPI flash
- Built-in CMSIS-DAP/J-link debug probe
- Ethernet, DMIC, SD card, USB HS/FS ports
- Stereo audio codec
- Arduino UNO R3 compatible expansion ports
- Shield board with TJA1059 dual transceiver
- Supported by MCUXpresso SDK for MCUXpresso IDE, Keil and IAR tools

*LPCXpresso54628 now also available, CAN-FD shield available separately*
LPC546xx Block Diagram

CPU Cortex-M4F
- LPC54628 up to 220MHz
- Other parts up to 180MHz

Memory
- Up to 512 KB Flash
- Up to 200 KB RAM
- 16 KB EEPROM

Interfaces for connectivity & sensors
- Stereo DMIC subsystem
- 1x HS USB (H/D) w/ on-chip HS PHY, XTAL-less FS USB (H/D)
- 10 SPI, 10 I2C, 10 UART, 2 I2S channels (max 10 channels total)
- Graphic LCD with resolutions up to 1024x768
- Ethernet with IEEE1722 timestamp
- 2 x CAN-FD controller (LPC5461x and LPC54628)
- Quad SPI flash interface
- External Memory interface (up to 32 bits)

Packages
- LQFP208 (28 x 28 mm), TFBGA180 (12 x 12 mm)
- LQFP100, TFBGA100

Operating
- Operating voltage: 1.71 to 3.6V
- Temperature range: -40 to 105 °C
Internal Bootloader
ISP: In-System Programming

- LPC546xx has various options to load code
  - USART/I2C/SPI
  - USB0/USB1
  - Plus programming via SWD debug port

- Per default, they are all enabled
  - Pulling ISP_PINx low on reset activates them

- They can all be disabled by software
  - If disabled by secondary, secure bootloader, then ISP/SWD can no longer be used
  - If keys are lost, no more updates...

- Default of our secondary bootloader
  - All remain enabled
Binary of bootloader (.hex) and all utilities required are available as free download

- [www.nxp.com/demoboard/om13094](http://www.nxp.com/demoboard/om13094) (download tab)
- [www.flashмагictool.com](http://www.flashмагictool.com)
- [Keepass.info](http://www.Keepass.info)
ESAcademy’s secure CAN bootloader
Free vs. Commercial

Free download
- Delivered as .hex
- On-chip ISP enabled
  - ISP remains as backdoor
- Fixed bit rate
  - 500/2000 kbps
- Fixed device and node ID
  - 15
- Pre-selected security methods
  - AES-GCM encryption and authentication
    www.cryptopp.com/wiki/GCM_Mode

License from ESAcademy
- Full C source code
- On-chip ISP may be disabled
  - No more updates if key is lost
- Configurable bit rate
  - Any combination supported
- Configurable device and node ID
  - 2-15 or 1-127
- Selectable security methods
  - All common methods supported
  - AES, SHA, RSA, EEC
Security limits

- **Keys are stored in regular Flash!**
  - Can be read by ISP/SWD
    - if not protected
  - Can be read from application
    - NOTE: Only manufacturer can load new application via secure bootloader

- **Key generation and installation?**
  - Keys must be „truly random“
  - Must happen in a trustworthy environment

- **Key storage**
  - Treat keys as valuable as what they protect
    - Here: source code
  - For small amount of keys, a password manager like „KeePass“ can be used
Secure Bootloader Security Review

- 3rd Party contracted with a security review of the secondary, secure CAN-FD bootloader

Work in progress, result expected within July
  - Result will be published in ESAcademy‘s Blog

MathEmbedded have many years of experience securing embedded systems for a wide range of global companies and in a number of market areas.

mathembedded.com
Where to get started

Files available from Monday 10\textsuperscript{th} of July 2017

- LPC range of MCUs at nxp.com/lpc
- LPCXpresso54618 board at nxp.com/demoboard/om13094
- CAN-FD driver add-ons under Downloads tab
- Free tools and software at nxp.com/mcuxpresso