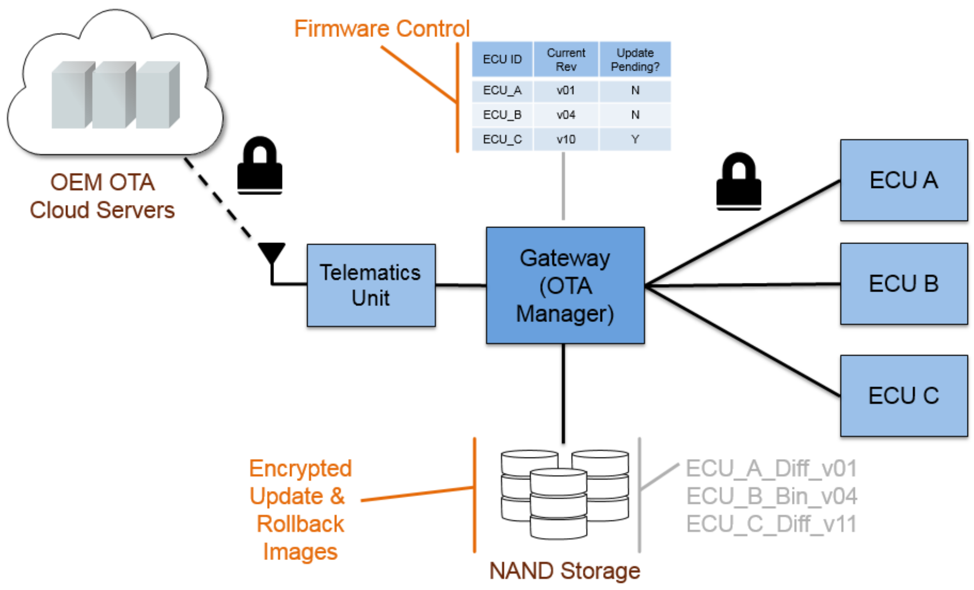
# Making Full Vehicle OTA Updates a Reality

Whitepaper

Author:

Daniel Mckenna
BU Automotive
NXP Semiconductors



| ECU ID | Current Rev | Update Pending? |
|---|---|---|
| ECU_A | v01 | N |
| ECU_B | v04 | N |
| ECU_C | v10 | Y |

-- This page is intentionally left blank –

## Contents

-- This page is intentionally left blank –

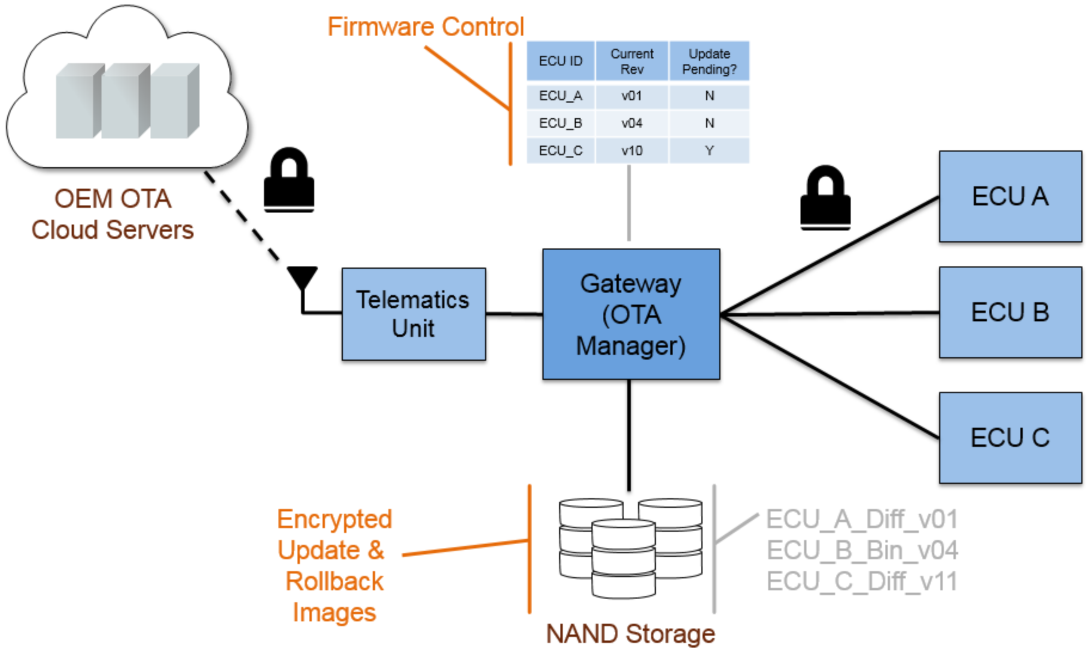
## Over-The-Air (OTA) updates for cars

The advantages of being able to perform in-the-field software updates to cars are well established: it will save manufacturers money, enable critical bugs to be patched immediately and allow compelling new features to be added to the vehicle at any time during its lifecycle.

At present however, few vehicles actually have the ability to receive updates. Even the cars that do support Over-The-Air (OTA) updates are only designed to update the infotainment or telematics systems. If an OEM discovers a fault in the firmware that controls a critical function such as the brakes or an airbag, then the vehicle has to be returned to the dealership. Electronic Control Units (ECUs) that control features like the engine, brakes, steering are deeper within the vehicle network architecture, and are typically Microcontrollers (MCUs) with small amounts of embedded flash and RAM. This brings constraints which require a different update approach.

Unlike a mobile phone or PC, car owners will not tolerate downtime of their vehicles while updates take place. Therefore, updates critical to vehicle operation should ideally take place seamlessly and invisibly in the background.

### Overview of Update Flow



| ECU ID | Current Rev | Update Pending? |
|---|---|---|
| ECU_A | v01 | N |
| ECU_B | v04 | N |
| ECU_C | v10 | Y |

The diagram above shows the key components which take the update file from the OEM's servers to the specific ECU within a vehicle. A secure connection is set up over a cellular network between an individual vehicle and the server. This allows the new, updated, firmware to be sent securely to the vehicle's Telematics Unit, and then on to the OTA Manager. The OTA Manager manages the update process for all ECUs within the vehicle. It controls the distribution of firmware updates to ECUs and will tell the ECU when to perform the update.

This is important in the case when multiple ECUs need to be updated simultaneously – e.g. to add a new feature to the vehicle which involves multiple ECUs. Once the update process is complete, the OTA manager will send confirmation to the OEM.

An external NAND flash can be fitted to the ECU which runs the OTA Manager to allow for the firmware update to be stored until they are required. The external flash can also be used to store backup copies of the firmware for other vehicle ECUs which can be called upon in the case of a major fault in an ECU update, which leaves an ECU without any working firmware. These backup copies would be secured via encryption and authentication protection to prevent any tampering of the firmware whilst stored in the external memory module.

The OTA Manager contains a table of every ECU within the vehicle including information such as serial numbers and current firmware version. This allows the OTA Manager to verify firmware updates which arrive and ensure that they are authorised for use in this vehicle. If the ECU being updated does not have security functionality then the OTA manager would also be responsible for decrypting and authenticating the incoming update.

## Update Process for Microcontroller ECU

When an ECU is scheduled to be updated, the new firmware update will arrive at the ECU in one of two formats:

- **Full binary:** The new firmware is sent in its entirety. This has the advantage of having no reliance on the previous firmware so that the update can take place even if the previous version has become corrupted. The two disadvantages of this method is the time taken to transmit the binary and the space required to store it at the receiving ECU. Many traditional ECUs are on CAN busses with typical speeds of 500kbit/s.

- **Diff file:** At the server, the OEM compares the new firmware to the previous version and creates a “diff” file which contains a list of differences between them. Depending on the amount of changes, this file will typically be <10% of the size of the original firmware making it quicker to transmit. This does mean that the diff is reliant on the previous firmware being a specific version.

Regardless of the format of the file, there are two methods for performing the update.

- **“A/B” approach:** This doubles the amount of flash on each ECU so that it can contain the current firmware in “primary” flash and has space for the full new version in the “secondary” flash. This approach is the ideal from an end user perspective as the ECU can remain in normal operation using the primary storage, while the new firmware can be written into the secondary storage in the background. Once the update is complete, the ECU can change to use the newly updated firmware (in the secondary flash blocks) at a convenient time (e.g. at next key-on or it can wait to synchronise the switch to occur with other ECUs being updated). As there is always a working firmware, there is no danger of putting the ECU into an inoperable state as it will always be possible to instantly “roll-back” to the previous working firmware. The obvious disadvantage is the cost of implementing two times the execution flash on the MCU.

- **<u>"In place" approach:</u>** In this case, only one version of the firmware exists on the device and individual blocks are erased and programmed as part of the update. The update cannot be run while the ECU is in normal operation – meaning that the vehicle will be inoperable for a period of time. This period of time will be determined predominately by the time required to reprogram the flash of the ECU. Major ECUs in the vehicle, like the engine controller, will have several MBs of flash which, should the full firmware need to be reprogrammed, can take in the order of tens of seconds to complete. This is a challenge for OEMs - will customers accept the fact that they are unable to start their engine for up to a minute?
  The added cost of the A/B method has to be traded off against the customer inconvenience caused by the "in-place" method. As only one version of the firmware exists in the "in-place" method, an error or reset during the update process can be tricky to recover from and if not carefully handled, could lead to the module (and hence the car) no longer functioning, making the vehicle inoperable until it is updated at a garage.

In the next chapter, I will cover how the OTA update process remains secure, the need for a central gateway, and the required microcontroller features to pull it all together.

## Securing the OTA update process

The ability to remotely update vehicle firmware opens up a new attack vector for hackers. The two main motivations for hackers will be to use the OTA mechanism to either reprogram critical ECUs to ultimately take control of the vehicle or to steal OEM firmware.

To prevent reprogramming, the authenticity and integrity of the firmware needs to be protected. This ensures that the firmware originates from a trusted source and that it has not been modified. There are several methods of implementing such protection (e.g. HMAC, CMAC, or a digital signature).

With authenticity checking of the firmware, an OEM can guarantee that only trusted firmware is used in an update process; however a hacker may still be able to read the plaintext (i.e. source binary) firmware to reverse engineer the source code, or steal data, which may for example lead to IP theft and/or privacy issues. Encryption of the firmware (e.g. using AES) can prevent this.

With integrity checking, the communication channel over which the firmware images are received will be protected to prevent commonly used “man-in-the-middle” attacks. Last but not least, the in-vehicle OTA manager which orchestrates the updates, should be protected against manipulation.

In summary, the following countermeasures shall be applied:

- Protect the authenticity and integrity of firmware to prevent a hacker from running modified code
- Encrypt the firmware to prevent a hacker from accessing code (IP) and data
- Establish secure end-to-end communication between the vehicle and OEM servers to prevent man-in-the-middle attacks.
- Ensure a secure, trusted location for the OTA Manager application

### Security Implementation

Typically the connection to the OEM server will be setup by the Telematics Unit. This unit can use Transport Layer Security (TLS), a proven standard widely used in online banking, to secure the communication over the mobile network.

The (secured) update file is then received by the OTA Manager which will have a table listing each ECU present on the vehicle along with the serial number and current firmware version number. This allows it to verify that the update is valid for this vehicle.

In addition, the update files should be end-to-end secured between the OEM server and the target ECU, as explained above. If the ECU to be updated supports firmware decryption and authentication, then the file can be sent as-is, from the OTA Manager to the target ECU. However, not all ECUs in the vehicle may have secure key storage and hardware accelerated security features. In this case the OTA Manager itself has to authenticate and decrypt the update on behalf of the target ECU and send the plaintext update, over the internal network to the target ECU (optionally an obfuscation method (where the code is deliberately made difficult for humans to understand) could be used to offer some level of protection). The lack of a hardware security module on an ECU thus leads to the update being sent unprotected over the in-vehicle network, which is not ideal. However for this to be hacked, physical access to the vehicle and a knowledge of which wires to access would be required in order to snoop the bus. If this is still considered too high of a risk then the module should not be updatable via OTA until the hardware is updated.

As mentioned above, the OTA manager should check whether an update is valid for the vehicle: not only to guarantee the consistency (compatibility) of all firmware images of the different ECUs in the vehicle, but also to prevent a hacker from being able to install an older firmware version in order to re-enable a patched exploit. This requires a secure method for storing the current firmware version number, which can then be checked against the version number of any received update.

The final mitigation strategy for OTA security is the location of the OTA Manager functionality. The OTA Manager is software that is orchestrating the updates of the firmware between the OEM servers and the vehicle's ECUs. If this is susceptible to being hacked remotely, a hacker potentially could have complete control of the vehicle. It is imperative that this is located somewhere with complete trust by the OEM, isolated from (potentially rogue) third party software. Some ECUs, such as the infotainment unit would be considered less trustworthy due to many third party apps (e.g. music, weather, navigation) being present, with potential backdoors to OEM firmware, making it a less desirable location for the OTA manager. The central gateway on the other hand, which sits central in the vehicle network (acting as the firewall for external and internal networks), typically only runs OEM trusted software and would be a strong location for the OTA manager.

## Need for Gateway to Manage OTA Process

As discussed in the previous section, the location of the OTA Manager greatly impacts how secure it can be. The ideal node within the vehicle which offers a good degree of security but still has the required connectivity to all in-vehicle networks is the gateway ECU.

By design, the gateway is the secured bridge between the many internal networks of the vehicle and the external networks of the outside world. The OTA manager is software which securely manages updates across the same internal to external bridge, therefore by running it on the gateway it would be able to carry out its job with no adjustments to existing network architecture.

The gateway offers isolation from the external interfaces a hacker may use to attempt to gain access. Compare this to a telematics or infotainment unit which can contain the interfaces listed below, providing an easy to access attack point for hackers.

- Cellular connections
- Wi-Fi
- USB
- CD/DVD

The software running on the gateway is reasonably compact, trusted auto-grade OEM code and a secure boot process will be used so that only firmware signed by the OEM can be executed on the device. Compare this with the huge code base running on an infotainment or telematics module which will contain multiple third party libraries, increasing the chances of a potential backdoor attack.

## Required Microcontroller Features

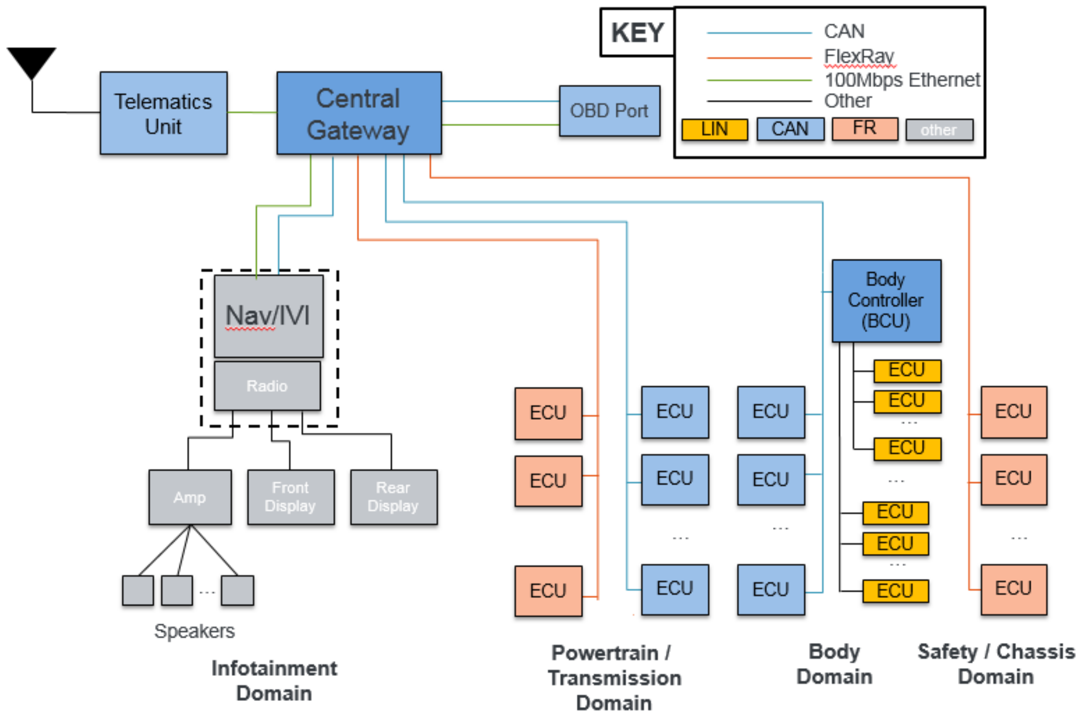
Almost all ECUs in a vehicle are capable of being updated using software mechanisms alone. However, the length of the update time and the lack of security will generally make this approach unacceptable. By using microcontrollers which provide specific hardware features, the updates can be processed faster, more securely and with less vehicle downtime.

**MCU Features:**

- **Read-While-Write flash:** Allows read access from a flash block while another block is being erased or programmed. Allows application to continue executing during update process.
- **Flash remapping/MMU:** Allows instant switching between new and old firmware images stored in different physical locations in flash. Note that flash remapping is preferred as MMU will only remap core accesses and not those from other bus masters such as the DMA.
- **Cryptographic Security:** Enables storage of private keys for decryption and authentication of incoming update.
- **Lifecycle Management:** Can be used to securely log the current firmware version and prevent illegal attempts to roll back to a previous version or to install unauthorised software.
- **Small code flash block sizes:** Having smaller flash blocks can improve update speed and efficiency. An in-place differential update will typically only modify a small number of locations. If, for example, these locations are in a 256K flash block then since the whole block has to be backed up then erased and reprogrammed, it will require more backup memory and more time to complete than if the location was in a 16K flash block.
- **Lockable flash regions:** Provides additional protection to critical code such as the bootloader during the update process to prevent accidental deletion.
- **Brownout Detection:** Ability to detect unexpected resets during update process. A reset during an update can leave the flash in an undefined state; the bootloader can then launch a recovery attempt upon exit from reset.
- **Multi-core:** A multi-core setup could be configured to allow one core to perform the update while the other is dedicated to running the existing application, minimising performance impact.
- **System Integrity Checking:** Grants the ability to detect and prevent system anomalies during an update.

## Example ECUs

A common network architecture for a vehicle is shown in the figure below.



Note that there are multiple different network protocols and these are all linked to the Gateway. Each ECU in the network is unique and will have attributes which affect how easily it can be updated, these include available CPU bandwidth, on-board security and RAM and flash size. It should be clear that a "one size fits all" approach is not feasible – a decision on which update and security method to use should be made on an ECU by ECU basis. A few potential use cases are shown below.

## Example ECU A

**Flash:** 2x internal flash available
**Security:** Supports CMAC authentication and AES-128 decryption
**Connection to Gateway:** FlexRay
**Vehicle Downtime:** none
**Security:** high

Since this ECU has double the required internal execution flash, the A/B method can be used for updates.

1. The updated firmware is trickle downloaded to the secondary flash blocks in the background whilst the current firmware version executes from the primary flash blocks. FlexRay is well suited to delivering trickle updates as the update can be sent using the lowest priority dynamic slot ensuring that it does not block any other required data.
2. The firmware is decrypted and integrity checked as it is downloaded. This module contains hardware acceleration for AES-128 decryption, CMAC authentication check and and secure private key storage. This allows for end to end encryption between the OEM's cloud servers and the ECU – keeping the data secured over the full network.
3. Once the new firmware is fully programed and authenticated, it can be activated when required with no vehicle downtime.

## Example ECU B

**Flash:** Has external NAND flash for local storage of new binary
**Security:** Supports CMAC authentication and AES-128 decryption
**Connection to Gateway:** CAN
**Vehicle Downtime:** long
**Security:** high

For this ECU a full new binary will be used for the update.

1. The encrypted binary will first be downloaded onto the gateway and will be stored in the gateways local NAND memory. The gateway can optionally authenticate and check the integrity of the update.
2. The gateway will then transmit the binary to the ECU as a background task whilst the vehicle is running. The ECU will store the encrypted update in external NAND flash.
3. Once the full binary has been stored in NAND flash the update can take place, this would need to occur during vehicle downtime. The ECU's internal flash would be erased and the decrypted contents of the NAND flash would be used to replace it.
4. After the process completes, the new firmware will be executed.

The addition of the low cost external NAND flash helps to minimise the vehicle downtime. Without it, the time taken to send to full binary over a slow CAN network would be added to the downtime.

## Example ECU C

**Flash:** Has at least 1 free block of internal flash
**Security:** No cryptographic Security
**Connection to Gateway:** CAN
**Vehicle Downtime:** long
**Security:** none

A diff file can be used in this case. The update would require vehicle downtime.

1. The encrypted diff file will first be downloaded onto the gateway and will be stored in the gateways local NAND memory.
2. As no cryptographic security exists on this ECU the diff file has to be decrypted in the gateway and sent on the CAN network as plaintext.
3. The diff file would be downloaded to the devices RAM and the update process would then be executed which will program and erase the necessary blocks as defined in the diff file. The additional flash block will be used to backup the contents of the flash block to be updated.

Ideally a copy of the ECUs firmware should also be stored on the gateway's external flash so that it can be used to perform a rollback should an unrecoverable error occur during the update process.

## Example LIN ECU

LIN is a low speed network typically running as 20kbit/s and used to network body electronics ECUs such as controllers for windows, mirrors and seats. The microcontrollers used on these ECUs are typically low end devices with small internal memories (e.g. 256K flash and 16K RAM). Diff style updates are typically not practical due to the lack of spare flash and RAM and the low performance CPU. An update could be carried out by breaking down the full binary and programming the device in chunks, but the transfer time over the LIN network is likely to take over 20seconds for a 256K device.

Although it is possible to perform OTA updates to ECUs on a LIN network, the large vehicle down time and software overhead required coupled with the fact that the smaller firmware size of these modules makes them less likely to require an urgent update, means that many OEMs will prefer to keep costs down and not enable OTA updates on these ECUs. As such any update would need to be carried out via a wired connection at a dealership.

## In Summary

The ability to perform OTA updates to ECUs within the vehicle is an essential feature. Given the wide range of different ECUs in the vehicle on many different types of networks there is no one update process which will work for all ECUs. The A/B method has a huge advantage from the driver's perspective as the update can be performed with zero vehicle downtime, however this can be a relatively more costly approach and is unlikely to work with existing legacy ECUs. The in-place update approach can be more cost effective but comes at the price of requiring vehicle down time. Each individual ECU should be evaluated to decide which strategy will lead to the ideal balance between cost and vehicle downtime.

The whole update process has to be managed by a central gateway ECU such as the NXP MPC5748G. This is used to keep track of firmware versions, distribute updates, synchronise firmware switching and to keep the whole process tightly secured.

## About the author

Daniel McKenna is a Systems and Applications Engineer at NXP Semiconductor. He has more than 10 years of experience working on Automotive Microcontrollers and Processors, from the initial definition stage through training and support for the final product.

## About NXP

-- This page is intentionally left blank –

-- This page is intentionally left blank –

www.nxp.com/automotivesecurity

Date of release: May 2016