# AES Bootloader

The AES bootloader will demonstrate how to perform secure firmware upgrades on the EFM32. The bootloader will allow users to upload encrypted firmware which will be decrypted, written to flash and run.

## Functionality

Out of reset the bootloader will first check a designated pin (DBG\_SWCLK).

If this is LOW (the default) it will start the application in flash. First it will check if there exists firmware at FIRMWARE\_START (firmware\_size != 0). If it does it will check the validity of the firmware by calculating a hash of the entire firmware and compare with firmware\_hash. If the firmware is valid execution will jump to this firmware. If not it will boot the initial firmware instead.

If DEBUG\_SWCLK is HIGH it will get ready to accept new firmware. The UART is initialized and the first character received is used to set the baud rate. The next character received will be interpreted as a command.

The only accepted command is ‘u’ which initiates a firmware upload. The bootloader will then respond with ‘C’ (capital C) to inform that it is ready to accept XMODEM-CRC packets.

The bootloader will then accept XMODEM packets. The first packet will include the firmware header. The size and hash values in this header is saved to flash in the Firmware Meta section. The size is also used to calculate the number of subsequent XMODEM packets to accept. Anything after this will be ignored. The packets will be decrypted and written to the firmware section of the flash on the fly. When the entire application has been written to flash, the bootloader will calculate a hash of the entire application and compare it against the supplied hash. If it matches the new firmware will be started.

## Flash Layout

The Flash will be divided into several sections. At FLASH\_START the bootloader will be placed. This first region will contain the bootloader code and the AES keys for both decryption and hash calculation.

At FIRMWARE\_META information about the uploaded firmware will be placed. This includes the size of the uploaded application and a hash of the entire application.

At INITIAL\_FIRMWARE the preprogrammed firmware will be placed. This firmware will be run if no other valid firmware is present.

At FIRMWARE\_START the uploaded firmware will be placed.

## Hash function

The hash function is a Cipher Feedback Mode (CFB) AES (with its own vector and key) where the last CipherText block is used as the application hash.

## Encryption

Encryption will use AES in either CBC, CFB or OFB. This will ensure that we can reuse the same (key, IV) pair for encrypting.

## Firmware file

When creating a firmware image a fixed-size header including the size and hash value is appended to the binary image before the entire file is encrypted.

## Debug Lock

The bootloader will clear the Debug Lock Word to protect the decrypted firmware and AES key. After debug access is locked, the only way to regain debug access is to use Debug Unlock which erases both Flash and SRAM.

## Firmware encryption tool

A tool for creating the encrypted images on a PC will be supplied with the bootloader. The tool will be distributed both in binary and source form. It will be written in C and use the libtomcrypt library (WTFPL license) to encrypt the firmware.

Optionally a tool for uploading the firmware can also be written. A separate evaluation to see if it is worth the effort will have to be conducted first.

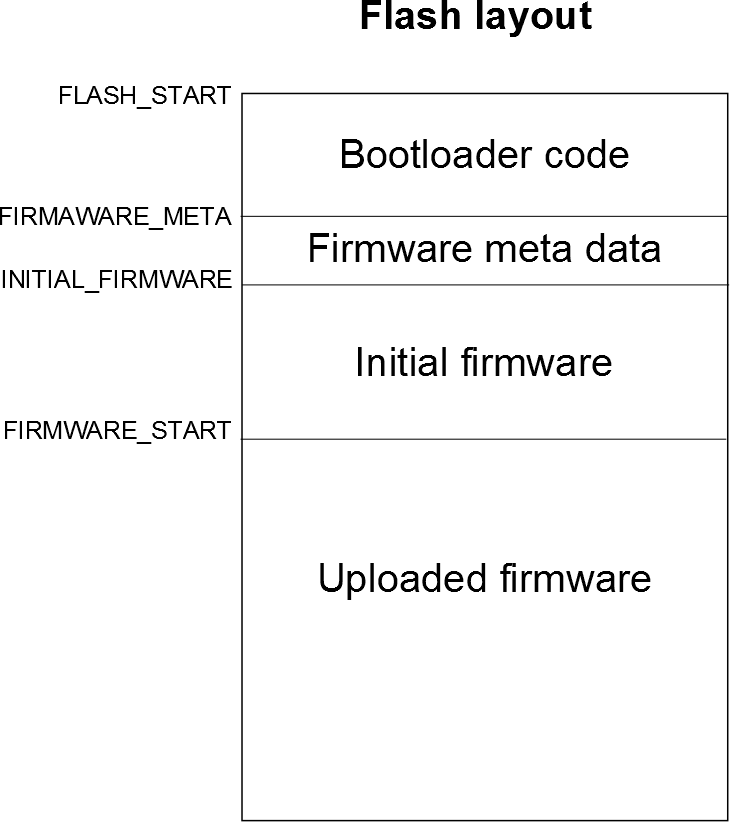
## Vulnerabilities

The bootloader will be vulnerable to attacks that either makes the software on chip read critical values from memory (such as the AES key) or are able to inject code into the application. The key and firmware will all be stored in plaintext in flash which both the bootloader and application firmware has read access to.

It should not be possible for an attacker to use the bootloader to upload his own program via the bootloader since he cannot create or modify both the application and hash value in such a way that the bootloader will accept it without knowing the encryption key.

No secret data must be placed in the User Data Page, since this page is not erased with debug unlock.



****

