User manual

DFU Signing

Experimental implementation

Features:

- DFU signing for application firmware images
- This feature has not been tested for bootloader or SoftDevice updates

Applications:

- Signing of application firmware images for DFU updates
- Verification of data from trusted source

All rights reserved.
Reproduction in whole or in part is prohibited without the prior written permission of the copyright holder.
July 9, 2015

Liability disclaimer

Nordic Semiconductor ASA reserves the right to make changes without further notice to the product to improve reliability, function or design. Nordic Semiconductor ASA does not assume any liability arising out of the application or use of any product or circuits described herein.

Life support applications

Nordic Semiconductor's products are not designed for use in life support appliances, devices, or systems where malfunction of these products can reasonably be expected to result in personal injury. Nordic Semiconductor ASA customers using or selling these products for use in such applications do so at their own risk and agree to fully indemnify Nordic Semiconductor ASA for any damages resulting from such improper use or sale.

Contact details

For your nearest dealer, please see www.nordicsemi.com

Main office:

Otto Nielsens veg 12
7004 Trondheim
Phone: +47 72 89 89 00
Fax: +47 72 89 89 89
www.nordicsemi.com

DNV
NORWEGIAN ACCREDITATION
No. S03
NS-EN ISO 9001 CERTIFICATED FIRM

RoHS statement

Nordic Semiconductor's products meet the requirements of Directive 2002/95/EC of the European Parliament and of the Council on the Restriction of Hazardous Substances (RoHS). Complete hazardous substance reports as well as material composition reports for all active Nordic Semiconductor products can be found on our web site www.nordicsemi.com.

Revision History

Date	Version	Description
July 2015	0.7	Initial release

Contents

1 Introduction

This document covers an experimental implementation of DFU signing.

The functionality and process of DFU signing is subject to change.

The DFU provides image verification by using Elliptical Curve Cryptography (in the following referred to as ECC) and the SHA-256 hashing algorithm.

NOTE: This is an experimental implementation and not intended for use in a final product until the functionality is out of experimental.

1.1 Prerequisites

Note that some prerequisites may be optional depending on method of doing the DFU Transfer.

Item	**Description**
nRF51 Development kit w/dongle	The dongle is optional.
Keil uVision 5.14 or later	IDE used to compile and program the experimental bootloader w/signing and the nrf_sec_module.
dfu_test_app_hrm_s110.zip	DFU image that can be programmed on the device and verified by signing. This project is the Heart Rate Monitor example using S110.
dfu_test_app_hrm_s110_forged.zip	Identical hex file as the previous, but with wrong signature.
Modified pc-nrfutil	Experimental build of pc-nrfutil that can be used to generate signing key, display verification key, and to create the DFU images.
Master Control Panel v3.9.0.6	Used for testing DFU transfer. Needs to be patched. (optional)
Android or iPhone	Used for testing DFU transfer. (optional)
nRFTools	Used for writing hex files and erasing the flash.

1.2 Resources

Following is a list of resources available in this experimental DFU signing:

Item	Description
bootloader_signing	Experimental project with bootloader that supports signing. Available as a Keil uVision project.
nrf_sec_module	Project that includes an open-source implementation to do SHA-256 hashing and ECC verify, accessible by SVC calls.
test_images	Signed DFU test images with correct/forged signatures.
pc-nrfutil.zip	Modified version of pc-nrfutil that supports key generation, signing, and packaging of DFU images using a new format.
nrf_sec.h	Header file that describes the SVC calls to access functionality in nrf_sec_module.

2 ECC signing/verification mechanism

2.1 Signing concepts

In the elliptical curve cryptography digital signature algorithm scheme (ECDSA), there are some concepts that differ in naming from the similar symmetrical cryptography methods, but the concepts should be shared.

2.1.1 SIGNING KEY

The signing key is equivalent to a private key. It is generated based on an elliptic curve. The signing key is not meant to be shared with anybody else.

Signing keys generated for the experimental bootloader w/verification are based on the elliptical curve called NIST P-256 described in RECOMMENDED ELLIPTIC CURVES FOR FEDERAL GOVERNMENT USE and other sources. This elliptical curve is also known as prime256v1 or secp256r1.

2.1.2 VERIFICATION KEY

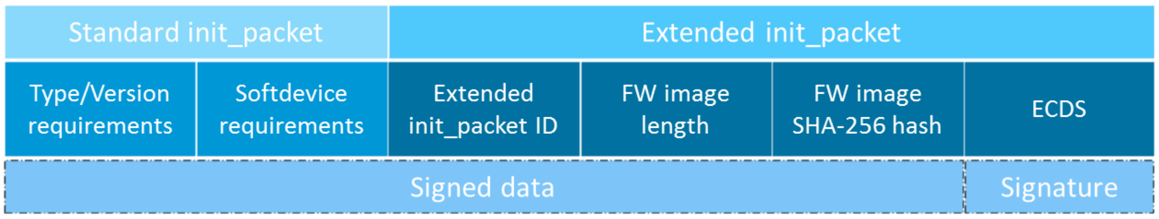
The verification key is equivalent to a public key. It consists of X and Y for a point on the elliptical curve, and it is used to verify the input data based on the signature. The verification key is calculated based on the signing key.

2.1.3 SIGNATURE

The signature is generated using a signing key and a hashing algorithm. For our experimental implementation of bootloader w/signing, we use SHA-256 as the hashing algorithm to generate the signature.

2.2 Signing strategy

The example project provides a template implementation of data verification based on signing of the DFU init_packet. This packet contains bootloader update restrictions and the length and a calculated hash digest of the firmware image.



Once the init_packet has been received by the target during the DFU transfer operation, the init_packet data (marked as “Signed data” in the image) will be verified based on the signature (ECDS). Verification is done using ECDSA with the elliptical curve NIST P-256 and the hashing algorithm SHA-256.

If the verification fails, no action will be taken to program the device and the DFU fallback behaviour will be activated. If the verification succeeds, normal DFU operation can continue.

From this point on, the DFU transfer works in the same way as previous versions of DFU, but with the added step of verifying the firmware image once transferred using SHA-256 instead of a CRC-16 checksum like in previous versions of DFU.

Since the experimental example supports dual-bank DFU operation, the application image will be downloaded in its entirety before the erase operations begins.

Dual-bank DFU operation is explained in greater details at the following address:

http://infocenter.nordicsemi.com/index.jsp?topic=%2Fcom.nordic.infocenter.sdk51.v9.0.0%2Fbledfu_memory_banks.html

3 Bootloader example

The experimental bootloader w/signing is provided as a Keil uVision project available at **<InstallLocation>\examples\experimental\bootloader_signing\pca10028\dual_bank_ble_s110**. Support for Keil Packs, IAR, and GCC will be added at a later stage.

Open the example by double-clicking the .uvprojx file. Compile it by pressing F7 or clicking **Rebuild**.

3.1 Prerequisites

The bootloader w/signing example requires the nrf_sec_module and the S110 SoftDevice to be programmed on the device.

3.2 Storage of verification key

This example implementation of the bootloader w/signing supports only one verification key. The verification key is hard-coded into the bootloader. To make the example project work with a DFU package signed by a signing key, the corresponding verification key must be in place. Any mismatch will lead to failure to verify.

In future releases of bootloader w/signing this verification key may be placed elsewhere in flash. In future implementations the bootloader w/signing may support more than one verification key.

3.3 LED assignments

The different states that the bootloader is in are signaled by the LEDs:

- LED 1: Advertising
- LED 2: Connected
- LED 3: Bootloader mode active

4 nrf_sec_module

The security module is provided as a self-contained project that can generate a hex file with functionality that is accessible by Supervisor calls (SVC calls). The nrf_sec_module provides the following methods:

- Generate hash digests from input data using SHA-256 hashing algorithm
- Verification of signed data based on input data, verification key, and an ECDS signature.

4.1 License

The ECC implementation provided with this release is provided under an open-source license. For the full license text and restrictions for use, modification, and similar, see the header file at **<InstallLocation>\examples\dfu\experimental\nrf_sec_module\ecc\ecc.h**.

4.2 Replacing the nrf_sec_module

It is possible to replace code in nrf_sec_module with an alternative implementation of ECC and the SHA-256 hashing functionality, as long as the signature of the SVC calls defined in nrf_sec.h is unchanged.

5 pc-nrfutil (experimental)

An experimental version of pc-nrfutil has been bundled with this release to facilitate signing key generation, verification key printing, firmware image signing, and packaging. This tool is required to generate DFU images (zip files containing everything that is necessary to set up for DFU transfer of an image from the host side).

5.1 Requirements

- Python (2.7.6 or newer, but not version 3. 32-bits)
- pip (https://pip.pypa.io/en/stable/installing.html)
- Python setuptools (upgrade to latest version: pip install --upgrade setuptools)
- Python modules listed in requirements.txt (run pip install -r requirements.txt)

This build of nrfutil relies on publicly available Python libraries that Nordic Semiconductor has no direct control or ownership over. Any claims to patent infringement, copyright infringement, and/or other legal liabilities fall on the end user and are not the responsibility of Nordic Semiconductor.

5.2 Setup

To set up pc-nrfutil, go into the folder **<InstallLocation>\examples\dfu\experimental**.

Unzip the file called pc-nrfutil .Use the following commands to install the utility as a Python package:

```
pip install --upgrade setuptools
pip install -r requirements.txt
python setup.py install
```

Use the following command to generate a self-contained Windows exe version of the utility:

```
python setup.py py2exe
```

The generated executable will be available in a subfolder called 0.2.3. This executable (nrfutil.exe) can be copied to a more suitable location on your hard drive or it can be run directly from that folder.

5.3 Commands

5.3.1 GENERATING THE SIGNING KEY

To generate a signing key, run the following command (example):

```
nrfutil.exe keys --gen-key c:\temp\priv.pem
```

This will generate a signing key called priv.pem in pem format at the given location. This signing key is generated using the elliptical curve NIST P-256.

It is possible to generate signing keys using OpenSSL, but doing so is beyond the scope of this document. When generating a signing key, it is vital that the elliptical curve NIST P-256 is used for generation. This curve is available under the alias **secp256r1** or **prime256v1**. Signing keys generated with any other curve will lead to invalid signatures.

It is vital that the private key is stored securely and with limited access. If the signing key is lost, it is impossible to reproduce it. Any effort to sign with a different signing key will cause the DFU operation to fail.

5.3.2 ADDING THE VERIFICATION KEY TO THE BOOTLOADER

The verification key is calculated from the signing key.

To print the verification key, run the following command (example):

```
nrfutil.exe keys --show-vk=code c:\temp\priv.pem
```

priv.pem is the input. The output will contain two variables Qx and Qy, similar to this:

```
static uint8_t Qx[] = { 0x39, 0xb0, 0x58, 0x3d, 0x27, 0x07, 0x91, 0x38, 0x6a,
0xa3, 0x36, 0x0f, 0xa2, 0xb5, 0x86, 0x7e, 0xae, 0xba, 0xf7, 0xa3, 0xf4, 0x81,
0x5f, 0x78, 0x02, 0xf2, 0xa1, 0x21, 0xd5, 0x21, 0x84, 0x12 };
static uint8_t Qy[] = { 0x4a, 0x0d, 0xfe, 0xa4, 0x77, 0x50, 0xb1, 0xb5, 0x26,
0xc0, 0x9d, 0xdd, 0xf0, 0x24, 0x90, 0x57, 0x6c, 0x64, 0x3b, 0xd3, 0xdf, 0x92,
0x3b, 0xb3, 0x47, 0x97, 0x83, 0xd4, 0xfc, 0x76, 0xf5, 0x9d };
```

These two variables must be copied and placed into the file **dfu_init_template_signing.c** replacing the previous definitions in the code.

5.3.3 SIGNING THE FIRMWARE IMAGE

Signing a firmware image requires a signing key. This key can be added using the argument --key-file when generating the DFU zip image (example):

```
nrfutil.exe dfu genpkg --application c:\temp\some.hex --key-file
c:\temp\priv.pem c:\temp\dfu_signed.zip
```

Running this command will convert the hex file (some.hex) to binary, hash it with SHA-256 and generate an init_packet that is subsequently signed (with priv.pem). The full content is then packed into a zip file (dfu_signed.zip) that can be programmed using different DFU transfer methods.

Note that the previous call generates a DFU image that does not provide any form of device/type or application version restrictions and no restrictions on updateable SoftDevice versions. To limit what kind of devices the generated DFU-packet can be programmed to, use any the following optional arguments:

```
--application-version
--dev-type
--dev-revision
--sd-req
```

These arguments are described in the original DFU documentation available online at the following webpage:

http://infocenter.nordicsemi.com/topic/com.nordic.infocenter.sdk51.v9.0.0/bledfu_example_image.html

6 Master Control Panel Patch

The Master Control Panel (and an nRF51 dongle) can be used to connect and bond to Bluetooth devices to inspect profile data and update characteristics. In addition, it supports over-the-air DFU operations.

The current release of the Master Control Panel, 3.9.0.6, does not support DFU with signing.

There is a patch available at **<InstallLocation>\ examples\dfu\experimental\master_control_panel_patch**.

This patch is required only for the PC version of Master Control Panel and works only with version 3.9.0.6.

Copy the files in this directory to:
%PROGRAMFILES(x86)%\Nordic Semiconductor\Master Control Panel\3.9.0.6\lib\dfu

It is possible to use other DFU transfer methods to program the device (like nRF Master Control Panel for Android or nRF Toolbox for Android or iPhone).

7 Testing the DFU signing functionality

The easiest way to test DFU signing is to use the precompiled and packaged images available at the following location:

<InstallLocation>\examples\dfu\experimental\test_images

In this folder, there are two DFU image zip files:

Item	Description
dfu_test_app_hrm_s110.zip	File that has been generated using the key priv.pem found in the same folder as the test image.
dfu_test_app_hrm_s110_forged.zip	File that has been generated with a different private key. This will fail during verification.

The folder also contains the signing key used to generate the valid DFU-image (priv.pem) and a precompiled version of the nrf_sec_module (nrf_sec.hex).

7.1 Prerequisites

To be able to test these files, the device must have been programmed with a valid S110 SoftDevice, nrf_sec_module, and bootloader w/signing. This is an example of a how to do this, given that you have all the relevant hex files:

```
nrfjprog --eraseall
nrfjprog --program <path_to_softdevice.hex>
nrfjprog --program <path_to_nrf_sec.hex>
nrfjprog --program <path_to_bootloader.hex>
```

It is also possible to do flash programming of **nrf_sec_module** and the **bootloader_signing** project directly from the Keil uVision projects.

After the device has been programmed, it must be put in DFU bootloader mode. This is indicated by LED 3 being on. To enter DFU bootloader mode, reset the device or hold down Button 4 and power-cycle the device.

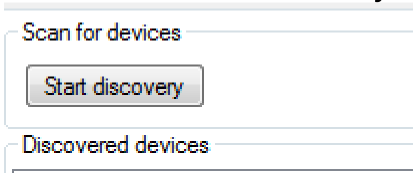
7.2 Verifying with master control panel

This method of verification requires an nRF51 dongle programmed with the master emulator firmware.

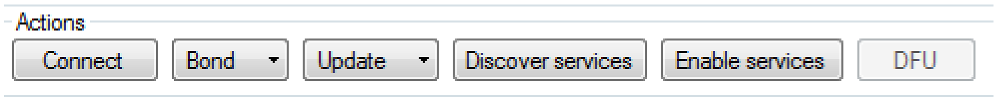
Verifying the image with the patched version of Master Control Panel is done in the following way:

1. Start the Master Control Panel. Select the COM port that corresponds to the correct Master Emulator device in the dropdown at the top.

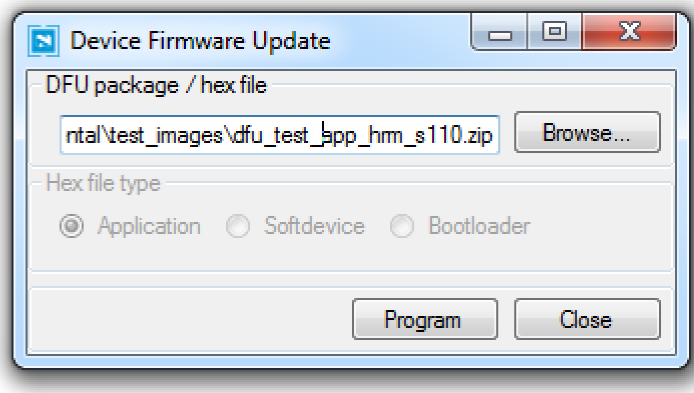
2. Click the button **Start discovery**.



3. From the list of discovered BLE devices, select the one named DfuTarg and click **Select device**.



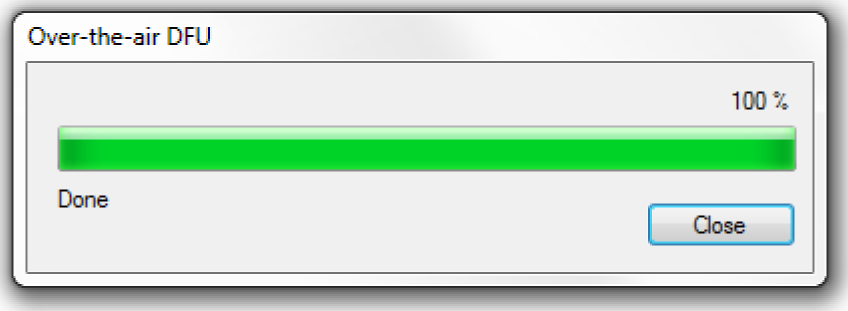
4. Press **Connect** and **Discover Services**. At this point, the **DFU** button should be enabled. Click the **DFU** button. A popup will be shown on the screen. Provide the path to a zip file from the test_images folder and click **Program**.



ntal\test_images\dfu_test_app_hrm_s110.zip

Hex file type: [✓ Application] Softdevice Bootloader

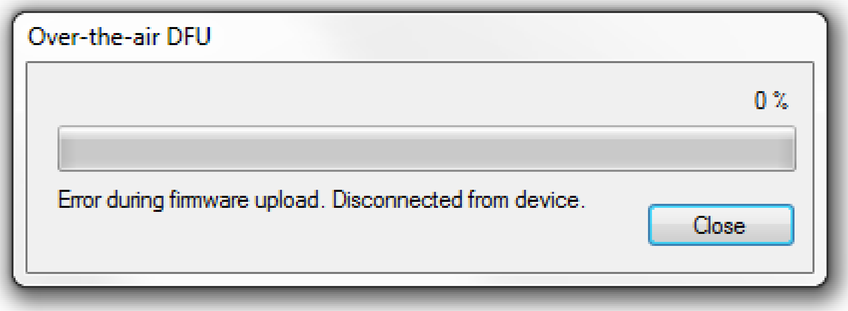
During flash operation a status window with a progress bar is shown. This process will be stalled for the duration of the verification, which takes approximately 30 seconds. After verification is successful, the progress bar should indicate that the data is being transferred to the device. If transfer is successful and if the hash has been verified, the following output will be shown:



100 %

Done

The device should also restart automatically. If on the other hand verification fails, the following will be shown:



0 %

Error during firmware upload. Disconnected from device.

5. If the transfer was successful, close the programming dialog and click **Back** in Master Control Panel to go to the startup screen. Right-click on the list of detected devices and press clear. Click on the button called **Start discovery**.

Your device should now be identified as Nordic_HRM (Hearth Rate Monitor).

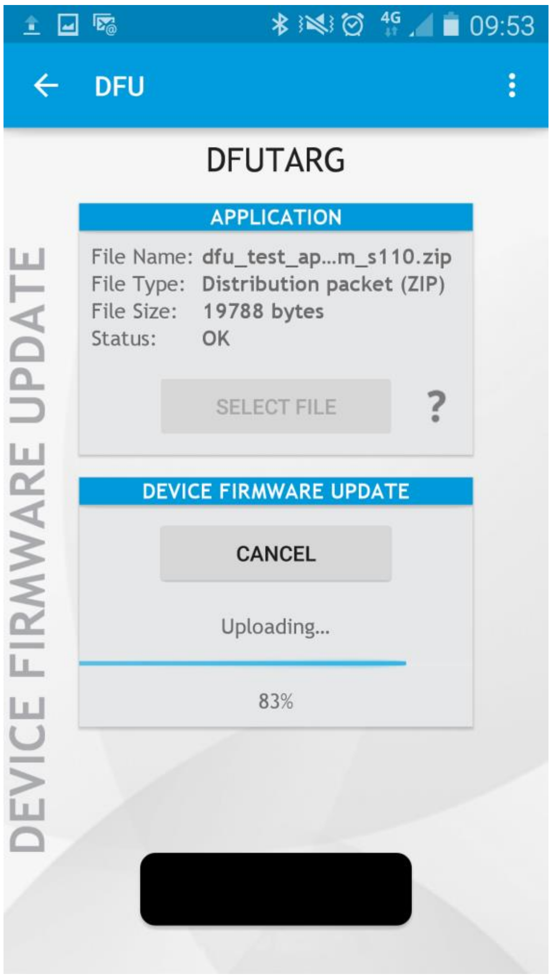
7.3 Verifying with a phone

It is possible to do over-the-air DFU using an Android phone or iPhone. This requires the nRF Toolbox application available from Google Play or iTunes.

In the nRF Toolbox app, click the icon for DFU programming.

Select the zip file image and the device that has the name **DfuTarg**. Click **UPLOAD**.

Progress will be shown during operation:

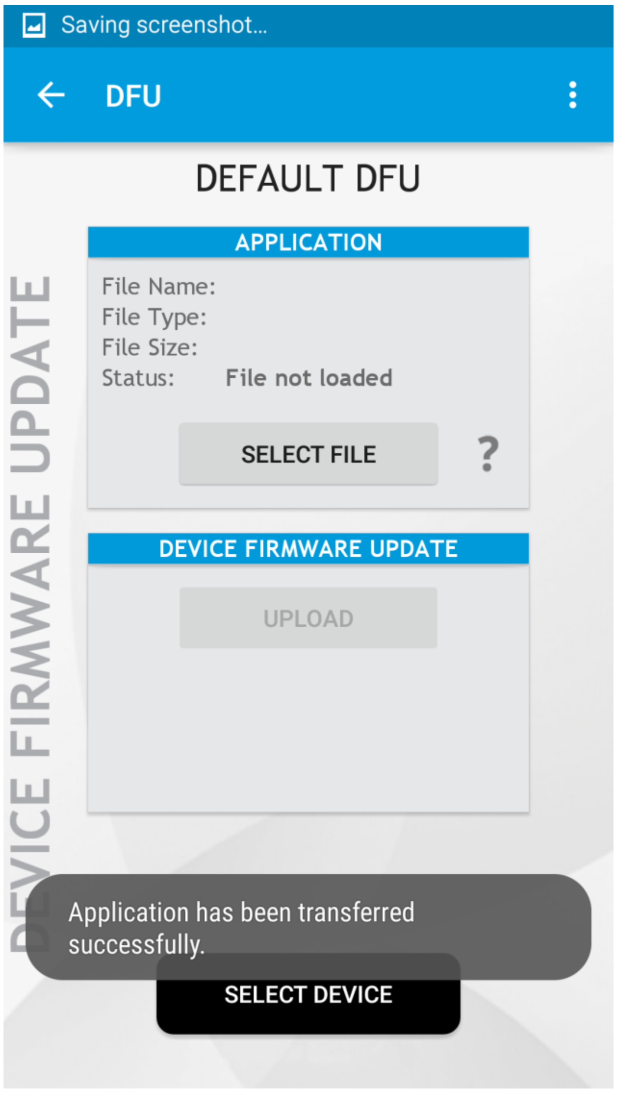


File Name: dfu_test_ap...m_s110.zip
File Type: Distribution packet (ZIP)
File Size: 19788 bytes
Status: OK

[redacted]

(Example from Android)

After a succesful transfer, the following screen will be shown:



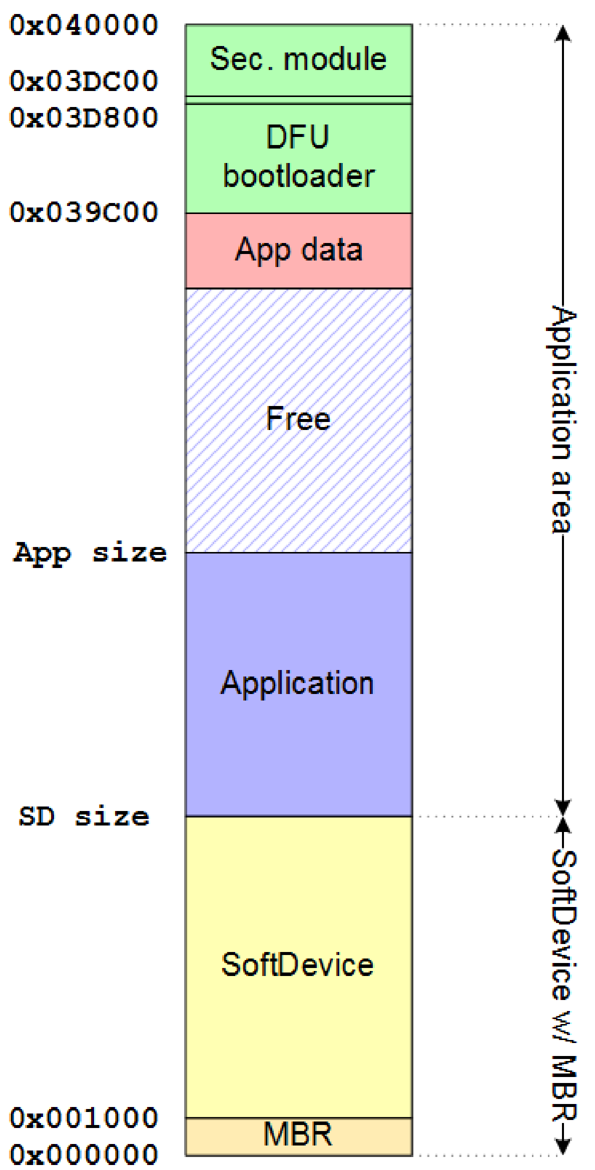
Application has been transferred successfully.

(Example from Android)

The device will restart if the update was successful. It should now be possible to utilize the HRM application in nRF Tools to verify that the DFU operation succeeded.

8 Memory layout

The bootloader w/signing is placed in memory directly underneath nrf_sec_module in the flash layout.



The last flash page between DFU Bootloader and nrf_sec_module is reserved for the bootloader to store bootloader-specific settings. This page starts at 0x3D800 and it is 0x400 long.

Besides the placement of the nrf_sec_module, flash memory layout during the different phases of the DFU transfer should be consistent with the original documentation available at http://infocenter.nordicsemi.com/topic/com.nordic.infocenter.sdk51.v9.0.0/bledfu_memory.html