Getting started with Nordic's Secure DFU bootloader, a step by step guide

Hung Bui 20 Mar 2017

Content of this blog based on the documentation of BLE Secure DFU example and Bootloader module library in our SDK. Here we provide you with step by step introduction to make it easier to follow and test Secure DFU.

The Secure DFU is the new DFU bootloader provided from nRF5 SDK v12. The old bootloader in SDKv11 and earlier is now called Legacy DFU. Secure DFU is not backward compatible with Legacy DFU.

What you would need to prepare before we start:

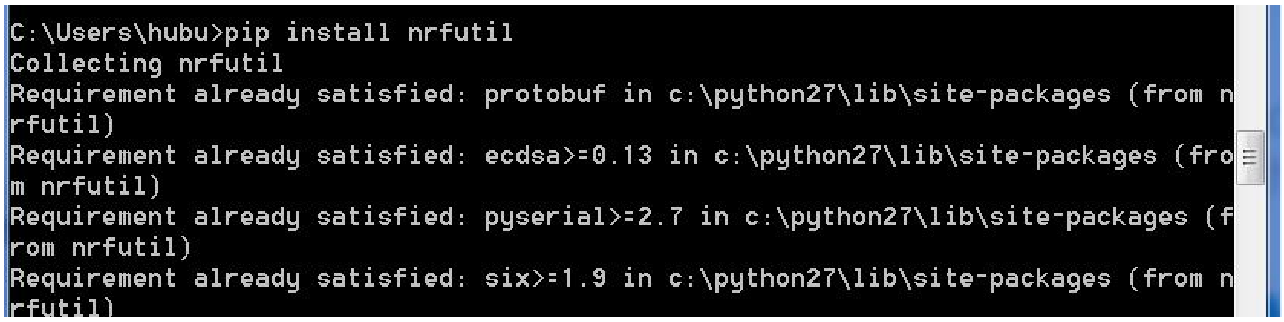
1. Up to date nRF5 SDK (minimum SDK v12)
2. Python with pip installed. See here if you don't have pip with Python.
3. Install version 4.9-2015-q3-update of the GCC compiler toolchain for ARM.
4. Make is also required (use MinGW, GNU Make, or Xcode)
5. A nRF5x DK or your own nRF5x board
6. A phone with BLE or a PC with an extra nRF5x DK or Dongle.

Step A. Generating keys

We need a pair of Public and Private Key to encrypt the signature to sign the DFU image using ECDSA_P256_SHA256.

Nordic provide nRFutil tool to generate these keys. nRFutil.exe can be downloaded from github here.

Or acquired from python using `pip install nrfutil`. To check for update, call `pip install nrfutil --upgrade` it.



```
C:\Users\hubu>pip install nrfutil
Collecting nrfutil
Requirement already satisfied: protobuf in c:\python27\lib\site-packages (from n
rfutil)
Requirement already satisfied: ecdsa>=0.13 in c:\python27\lib\site-packages (fro
m nrfutil)
Requirement already satisfied: pyserial>=2.7 in c:\python27\lib\site-packages (f
rom nrfutil)
Requirement already satisfied: six>=1.9 in c:\python27\lib\site-packages (from n
rfutil)
```

A1. Generate your own private key. Type this command line:

```
nrfutil.exe keys generate private.key
```

private.key file will be generated. You will keep this key secret

A2. Generate your public key based on your private key.

```
nrfutil keys display --key pk --format code private.key --out_file public_key.c
```

After you have the public_key.c file we can move to the next step, build the bootloader.

Step B. Build the bootloader

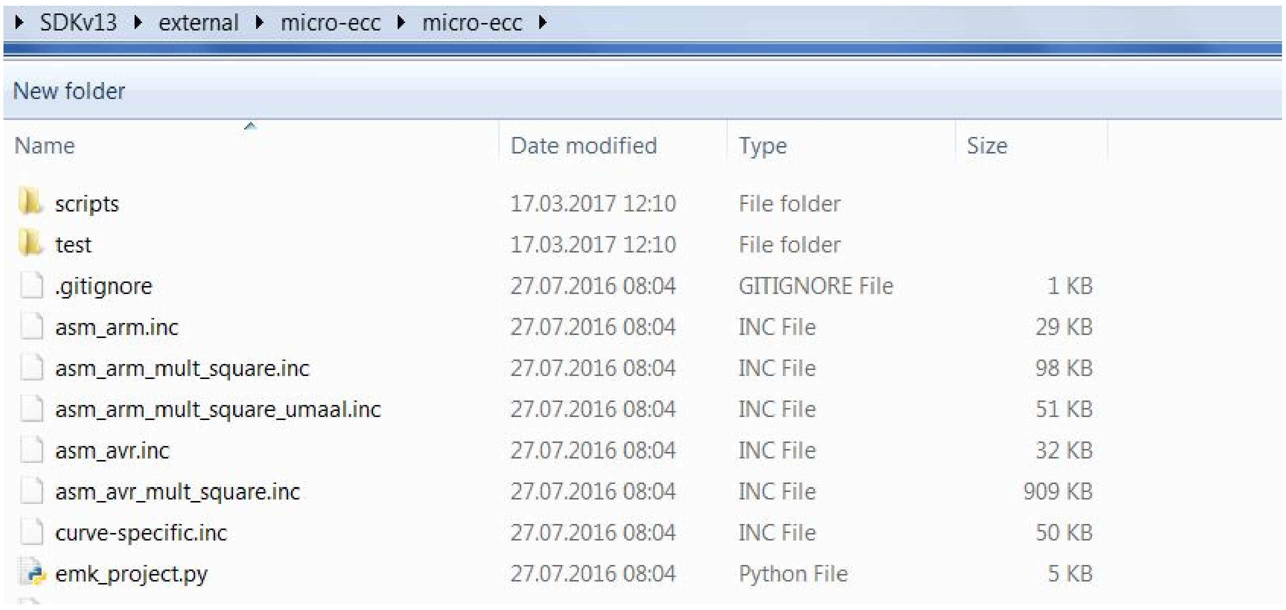
The bootloader is located at \examples\dfu\bootloader_secure_ble

Note that there are 2 flavors. There is one with _debug surfix. You use this _debug version if you want to avoid version checking, we will come to it later. For now, we will use the normal bootloader

B1. Compile the uECC library.

uECC library is needed for the bootloader to decrypt the signature. uECC is an external library and have to be downloaded from github here. Note: there is a license requirement comes with it.

You have to clone/download the library to the SDK folder: `SDKFolder\external\micro-ecc\micro-ecc` . It should look like this:

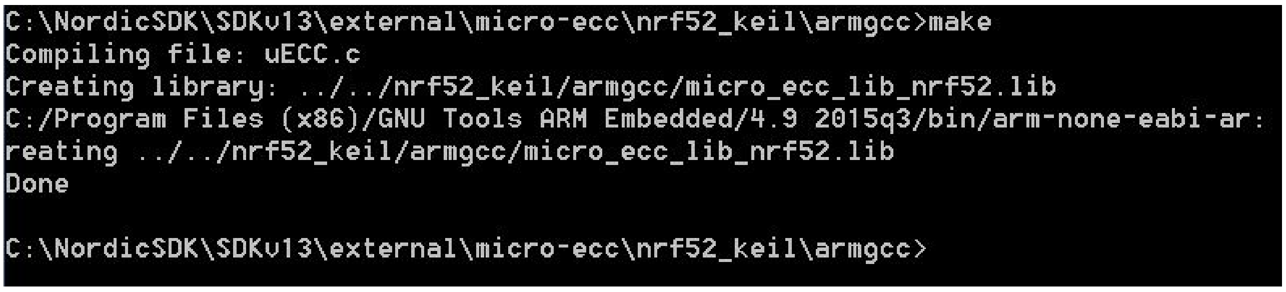


▸ SDKv13 ▸ external ▸ micro-ecc ▸ micro-ecc ▸

New folder

Name	Date modified	Type	Size
scripts	17.03.2017 12:10	File folder	
test	17.03.2017 12:10	File folder	
.gitignore	27.07.2016 08:04	GITIGNORE File	1 KB
asm_arm.inc	27.07.2016 08:04	INC File	29 KB
asm_arm_mult_square.inc	27.07.2016 08:04	INC File	98 KB
asm_arm_mult_square_umaal.inc	27.07.2016 08:04	INC File	51 KB
asm_avr.inc	27.07.2016 08:04	INC File	32 KB
asm_avr_mult_square.inc	27.07.2016 08:04	INC File	909 KB
curve-specific.inc	27.07.2016 08:04	INC File	50 KB
emk_project.py	27.07.2016 08:04	Python File	5 KB

Next step, inside `SDKFolder\external\micro-ecc\` choose the IDE and compiler that you want to use to build the bootloader. In my case I chose `SDKFolder\external\micro-ecc\nrf52_keil\armgcc` and type `make` to start building uECC.



```
C:\NordicSDK\SDKv13\external\micro-ecc\nrf52_keil\armgcc>make
Compiling file: uECC.c
Creating library: ../../nrf52_keil/armgcc/micro_ecc_lib_nrf52.lib
C:/Program Files (x86)/GNU Tools ARM Embedded/4.9 2015q3/bin/arm-none-eabi-ar:
reating ../../nrf52_keil/armgcc/micro_ecc_lib_nrf52.lib
Done

C:\NordicSDK\SDKv13\external\micro-ecc\nrf52_keil\armgcc>
```

B2. Copy the public_key.c file generated at step A2 to your bootloader folder, or a folder of your choice and include it into your project. Remove the dummy `dfu_public_key.c` in the project because it's replaced by public_key.c

Now you are ready to build the bootloader.

B3. Build the bootloader. Click build. If you have done step B1, B2 correctly, the bootloader should build successfully.

Note that every time you generate a new pair of public&private key, you need to update the public_key.c file and rebuild the bootloader.

Step C. Generate DFU .zip packet.

A DFU .zip packet is required for the DFU master to send new image file to the DFU target. The .zip file contains the image hex file(s) we want to update, the init data and the signature of the packet. In the main part of this tutorial, we only do application image update. For bootloader and softdevice update, please see the Appendix.

C1. Prepare the application hex file. Build your application and find the .hex file inside _build folder. Usually it's named `nrf52832_xxaa.hex` if you compile a SDK's example. Flash the application hex and verify it works normally without the bootloader (flash softdevice if needed).

C2. Generate the .zip file. Copy the **private.key** generated in step A1 to the same folder with your .hex application file.

In my case I use this script to generate the .zip file:

```
nrfutil pkg generate --hw-version 52 --application-version 1 --application nrf52832_xxaa.he
```

Explanation:

--hw-version: By default this should match with your chip. If you use nRF51 chip you should use "51". If you want to have your own hw-version code, you can define your in the bootloader, using `#define NRF_DFU_HW_VERSION your_hw_number`

--application-version: By default the start number for application version is 0. To be able to update new application, the application version should equal or greater than the one the bootloader stored. This case I use 1. Read more about version rule.

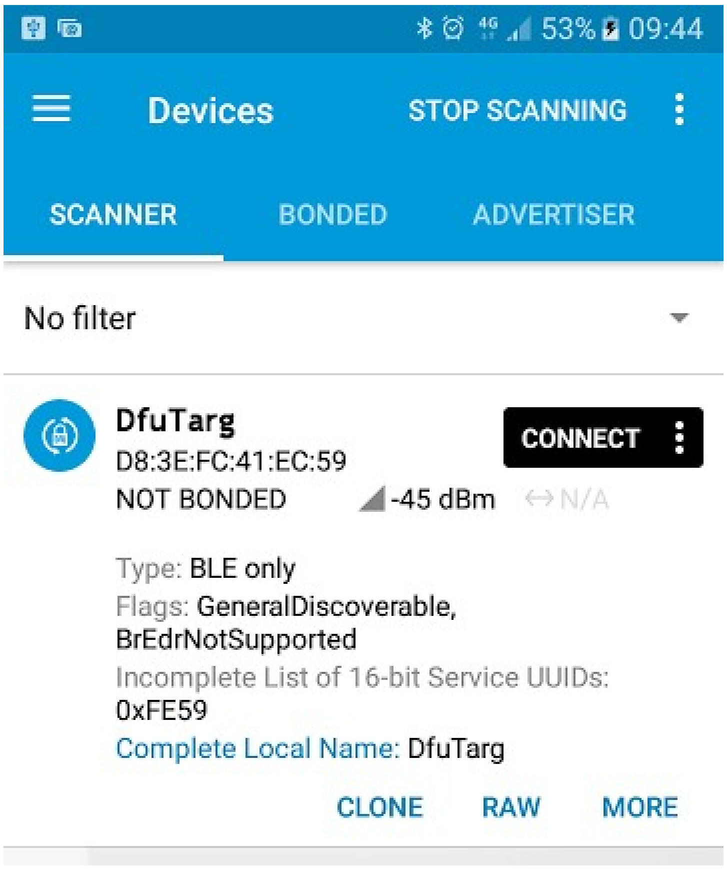
--sd-req: In my case my application runs with Softdevice S132 v4.0.2. The code number for this softdevice version is **0x98**. You can find the softdevice code number by typing `nrfutil pkg generate --help` . If it's not in the list just yet, you can find the code using nRFGo Studio, just open any board that contains the softdevice.

--application : Tells nrfutil that you going to update the application the the application image is provided.

Step D. Test DFU

Now you have your DFU .zip file and the bootloader ready, it's time to actually do DFU.

D1. Flash bootloader and softdevice. Use nRFGo Studio or nrfjprog or the IDE to flash the softdevice first, then the bootloder. Verify the bootloader start advertising as "DFUTarg".



DfuTarg
D8:3E:FC:41:EC:59
NOT BONDED -45 dBm ↔N/A

Type: BLE only
Flags: GeneralDiscoverable, BrEdrNotSupported
Incomplete List of 16-bit Service UUIDs: 0xFE59
Complete Local Name: DfuTarg

D2. Copy the DFU .zip file you just generated in C2 to the phone or to the folder on PC of your choice.

D3. Use nRFConnect/nRFToolbox app either on the phone or on PC to connect and OTA DFU using the .zip file you copied.

D4. Verify that the new application runs after the DFU process reaches 100%, for example it starts advertising with the name that you set in your application. If you can see that, congratulation you have done your first Secure OTA DFU!

Step E. Update new application firmware (Optional)

E1. Generate new firmware, for example modify the advertising device name, or the LED indication.

E2. Generate new DFU .zip package. Update new application version (at least equal or higher original version)

E3. Switch the device to DFU mode, do DFU update and verify new image is runnning.

Appendix 1. Advanced features

1. Combine application, bootloader, bootloader setting and softdevice image

The bootloader uses its bootloader setting to detect if a valid application is flashed on the chip or not. CRC check will be performed when booting. If we simply flash the application with the bootloader using the programmer, the bootloader won't be able to detect there is a valid app already flashed and it will enter DFU mode, your application won't start.

Only after a successful DFU update the bootloader would write to the bootloader setting to mark the valid app. But in production for example, you may not want to install the application by doing OTA DFU for thousands of device which takes lots of time. We can generate the bootloader setting manually and merge it with the bootloader to trick the bootloader to accept the pre-flashed application. We can use nrfutil to generate the bootloader setting and use mergehex.exe tool to merge the hex files. If you don't want to merge you can flash the setting first then flash the bootloader.

The script to generate bootloader setting is

```
nrfutil settings generate --family NRF52 --application yourApplication.hex --application-ve
```

After this command the bootloader_setting.hex will be generated. It includes the application version and the CRC that match with the yourApplication.hex provided. This should be used to overwrite the default bootloader setting in the bootloader.

--bl-settings-version: For SDK12 and SDK13 we only have same bootloader setting version = 1

--application-version and --bootloader-version: The initial version of application and bootloader of your choice. In this case I chose 0.

--family: Device family, match with your chip. If you have nRF52840 you have to use NRF52840 instead of NRF52.

After that you can use merhex to merge bootloader with bootloader_setting.hex and additionally with the application and the softdevice if you want.

Usage:

```
mergehex --merge hexfile1.hex hexfile2.hex --output output.hex
```

Note: If your application requires UICR to be written, it's good idea to use this merge to write the application via a programmer instead of OTA. When you do OTA DFU the bootloader won't write to UICR.

After we have the merged hex, we can use one single hex to flash multiple devices and don't have to do OTA DFU to flash the application.

2. Buttonless DFU

The easiest way to enter DFU bootloader mode is to hold the Bootloader Button (button 4 on the DK) and reset the device. But there are chances that your device doesn't have any button or you just want to switch the application to bootloader without any physical contact. If it's the case, you need buttonless DFU, simply telling the application to switch to the bootloader via a BLE packet.

We provide the DFU Buttonless example at this folder `\examples\ble_peripheral\experimental_ble_app_buttonless_dfu`

You can follow the link to know how to test it. Here we try to explain how it works. The buttoless example is no different from normal application. You still needs to flash the bootloader. It's important that you make sure the bootloader works fine with normal application first before we start with the buttonless example.

The way it works is pretty simple, to switch, we write to the retention register GPREGRET a flag (BOOTLOADER_DFU_START = 0xB1) and then we do a soft reset. This is done in bootloader_start() in ble_dfu.c file.

Since the retention register keeps its value after the reset, the bootloader will check this value when it booting up after the reset and then can enter DFU mode instead of starting the normal application. This is the same as when we hold Bootloader button and trigger a reset.

Note that on SDK v12 we write to bootloader setting, instead of write to GPREGRET register. I think writing to GPREGRET is a cleaner way of doing this.

Next we find when bootloader_start() is called.

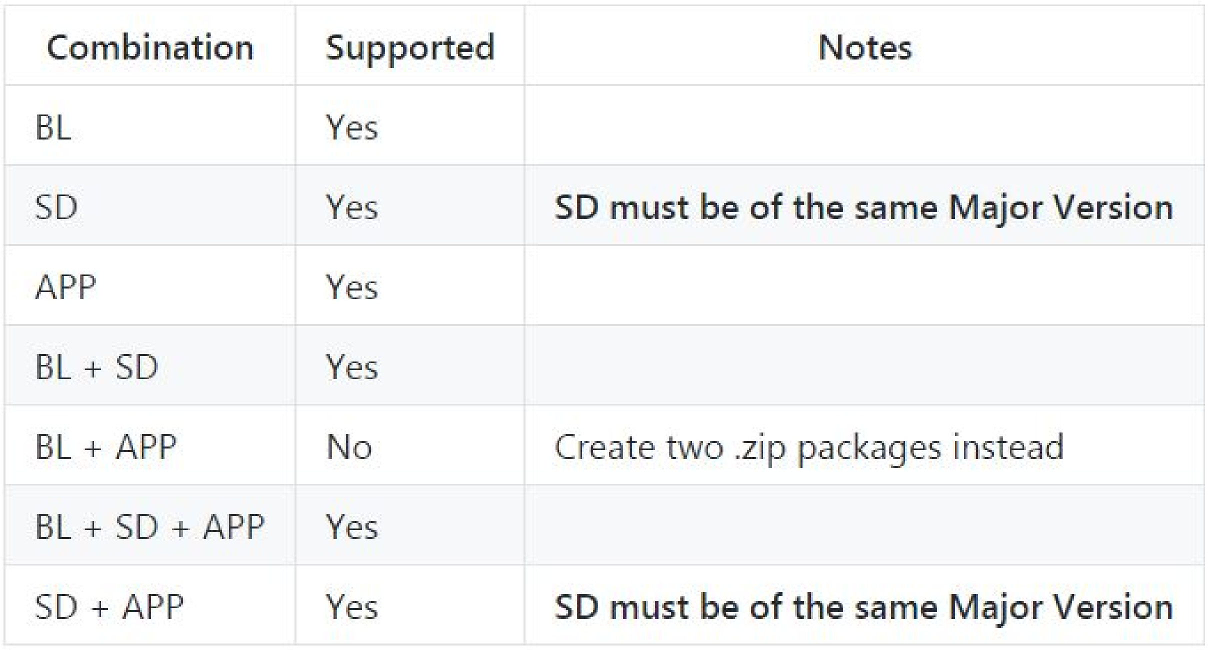
From SDK v13 we introduced DFU buttonless characteristic. The characteristic has indication and write properties. What we need to do is simply enable indication on the characteristic and write 0x01 to it. The Buttonless device will send an indication and wait for acknowledge from the DFU master. When the acknowledge from the central comes, it will call bootloader_start() (check on_hvc() function in ble_dfu.c). The connection will be dropped after that.

By the time of writing this blog, the experimental_ble_app_buttonless_dfu doesn't do any bond forwarding. So you may experience an issue if bonding is exist between the phone and the device. Please have a look at section **4. Bond forwarding** bellow.

3. Update softdevice and bootloader

It's possible to update the softdevice, the bootloader and the application also we support updating the combination of these 3 images. However, not all of the combinations are possible.

The table below summarizes the support for different combinations (it's from nrfutil documentation here).



Combination	Supported	Notes
BL	Yes	
SD	Yes	**SD must be of the same Major Version**
APP	Yes	
BL + SD	Yes	
BL + APP	No	Create two .zip packages instead
BL + SD + APP	Yes	
SD + APP	Yes	**SD must be of the same Major Version**

BL: Bootloader

SD: SoftDevice

APP: Application

The following code will generate a softdevice + application combination:

```
nrfutil pkg generate --hw-version 52 --application-version 1 --application application.hex
```

4. Bond forwarding TBD

If you have question related to this blog, please create a case in our forum with a link to this.

9 comments 0 members are here

kamisen 12 months ago
When the application has WDT, it will affect bootloader. Could you give some advisement?

Hung Bui 12 months ago
@kamisen: we will continue discussing this in your question.

Andrew_Engineer 10 months ago
Hello Hung,

I am trying to follow your instruction and get stucked at the step B1: Next step, inside SDKFolder\external\micro-ecc\ choose the IDE and compiler that you want to use to build the bootloader. In my case I chose SDKFolder\external\micro-ecc\nrf52_keil\armgcc and type make to start building uECC.

I have installed GCC compiler and MinGW, when I try to run "make" command from the DOS prompt I am getting, error message: "'make' is not recognized as an internal or external command, operable program or batch file."

I did search both directories "C:\MinGW" and "C:\Program Files (x86)\GNU Tools ARM Embedded\5.4 2016q3" and didnt see and "make" executable file.

Can you please double check and advise...

Thank you,

Andrew.

Martin 10 months ago

@andrew_engineer: For mingw try mingw32-make. This should do it.

Lukas Simma 9 months ago

I have download "Complete package, except sources" from gnuwin32.sourceforge.net/.../make.htm After unpacking i copied the three file from bin folter to armgcc\4.9_2015q3\bin and to armgcc\4.9_2015q3\arm-none-eabi \bin

With this commands in a script file i compiled the needed lib file:

```
@echo off
SETLOCAL
set PATH=C:\devel\nrf52\nRF5_SDK_13_0_0\external\armgcc\4.9_2015q3\bin;%PATH%
cd C:\devel\nrf52\nRF5_SDK_13_0_0\external\micro-ecc\nrf52_keil\armgcc
make
```

pause

Load Next