Inter-CESTI challenge: the WooKey use-case

WOOKEY PROJECT, ANSSI – Agence Nationale de la Sécurité des Systèmes d'Information

1 INTRODUCTION

This document is a global quick start for the use-case submitted to the Inter-CESTI 2019 Challenge.

All the source code of the WooKey project is online in the form of many repositories at the following URL: https://github.com/wookey-project/. A dedicated detailed documentation associated to the project and its repositories can be found here: https://wookey-project.github.io/.

2 PROJECT OVERVIEW AND RATIONALE COMPARED TO THE SECURITY TARGET

The WooKey project implements all the concepts presented in the Security Target, namely:

- *An isolation microkernel* focused on security and written in Ada with some parts proven in SPARK. The microkernel is *EwoK* and can be found here: https://github.com/wookey-project/ewok-kernel. Very detailed information about the internals and the syscalls can be found in the dedicated part of the project online documentation https://wookey-project.github.io/ewok/index.html.
- *A nominal mode* that is represented by dedicated *applications* that are isolated on top of the EwoK microkernel. The nominal mode presents a *mass storage USB device* on the PC host side. Associated tasks are:
  - The *USB task* that handles SCSI mass storage with the host PC (https://github.com/wookey-project/app-wookey-usb).
  - The *SDIO task* that manages the SD cards storage backend (https://github.com/wookey-project/app-wookey-sdio).
  - The *smart task* that interacts with the authentication AUTH token and handles sensitive elements (https://github.com/wookey-project/app-wookey-smart).
  - The *crypto task* that handles encryption and decryption of user data through DMA transfers, sitting between the USB and SD tasks (https://github.com/wookey-project/app-wookey-crypto).
  - The *pin task* that manages the user interactions through a dedicated GUI on the touchscreen (https://github.com/wookey-project/app-wookey-pin).
- *A DFU mode* used to the *secure updates* of the platform. This mode is entered voluntarily by the user through a dedicated physical button at boot of the platform. A DFU class USB device is then presented to the PC host. The associated tasks are:
  - The *dfu-USB task* that handles the DFU automaton with the host PC (https://github.com/wookey-project/app-dfu-usb).
  - The *dfu-flash task* that manages the STM32 internal flash storage backend for internal firmware update (https://github.com/wookey-project/app-dfu-flash).
  - The *dfu-smart task* that interacts with the DFU token and handles sensitive elements (https://github.com/wookey-project/app-dfu-smart).

Author's address: WooKey project, @team:wookey@ssi.gouv.fr, ANSSI – Agence Nationale de la Sécurité des Systèmes d'Information.

- The *dfu-crypto task* that handles decryption of firmware chunks through DMA transfers, sitting between the USB and flash tasks (https://github.com/wookey-project/app-wookey-crypto).
- The *pin task* that manages the user interactions through a dedicated GUI on the touchscreen (https://github.com/wookey-project/app-wookey-pin).

All the tasks use various *libraries* that are platform independent (abstracted from the underlying hardware) as well as dedicated drivers that are adherent to the WooKey platform (STM32F439, screen, and so on). All the software components should have logical dependencies between each other, from high level abstracted automatons to hardware dependent modules.

Libraries are for instance cryptographic algorithms (Elliptic Curves, AES, hash, etc.), protocol automatons (ISO7816-3 for the smartcard, SD for the SD cards, SCSI for mass storage, DFU for firmware update, etc.), various helpers (standard library, User Interface GUI, etc.). More details on WooKey libraries can be found here: https://wookey-project.github.io/libs.html. Drivers implement the low level part of protocols (SDIO for SD, USB endpoints handling, USART line for ISO7816-3, touch screen, etc.). Details on the drivers are given here: https://wookey-project.github.io/drivers.html.

A dedicated cryptographic architecture has been developed for the WooKey project, making extensive use of *secure elements* in the form of *Javacard smart cards* in order to authenticate the legitimate user (through a set of various PIN codes to limit rogue tokens), and perform a mutual authentication with the WooKey device. The user authentication AUTH, DFU and signature SIG tokens software can be found here: https://github.com/wookey-project/javacard-applet. Three Javacard applets (AUTH, DFU and SIG) are implemented, and more details about them (i.e. supported APDUs and so on) can be found in this section of the online documentation: https://wookey-project.github.io/javacard/index.html.

Beyond the software architecture, the hardware design has been made open hardware and can be found here: https://github.com/wookey-project/hard-wookey-motherboard for the main board, and here: https://github.com/wookey-project/hard-wookey-screen for the screen board.

3 HARDWARE DELIVERY

In order to facilitate the challenge process, we provide all the hardware necessary to test the WooKey project. The delivery is composed of:

(1) **A functioning WooKey device** (see Figure 1 and Figure 2) composed of a main board PCB, a screen PCB as well as the touch screen.
The front side of the WooKey PCB (Figure 1) presents the following noticeable elements:
   - An USB HS (High Speed) connector to be connected to the host PC. This is the main WooKey connection as a USB device (i.e. the link used to present itself as a mass storage or DFU device). As a side note, a USB FS (Full Speed) is present as WooKey can also be used as a Full Speed device: this connector is here as an optional feature. In the following, and depending on you WooKey configuration profile (used for firmware compilation), please connect either the USB HS or FS connector to your PC.
   - An SD card slot where an SD card is provided and already present (replaceable at your will).

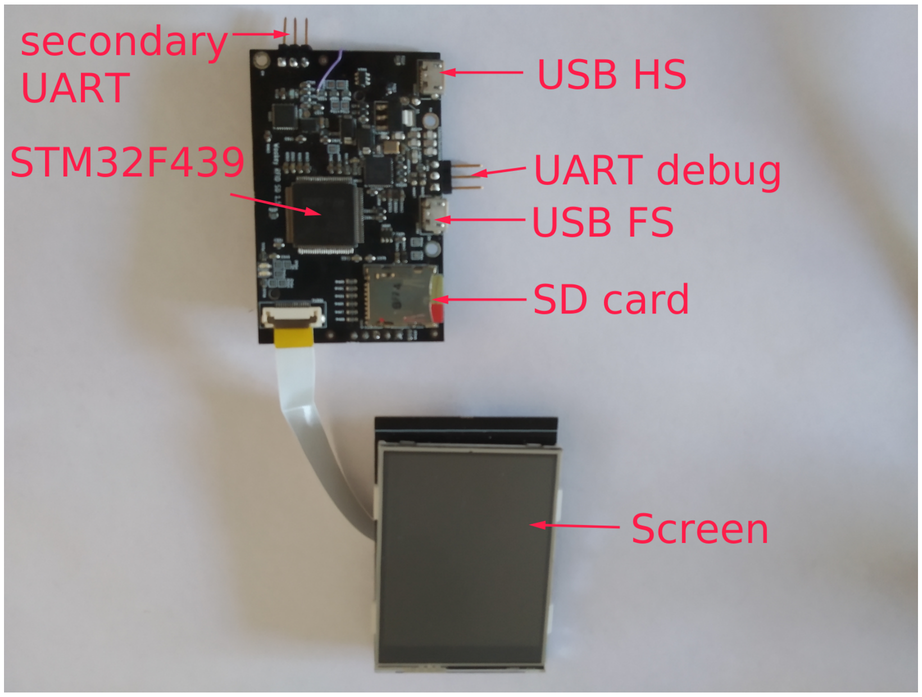

Fig. 1. WooKey main PCB (front), screen PCB and screen

- The main UART debug port (with three connectors corresponding to the classical RX/TX/GND). There is also a secondary UART connector that can be used by WooKey tasks developers. For following EwoK and tasks logs, you will have to connect/use the main UART debug port.

For obvious simplicity and reviewing reasons, the initial board provided is an **open WooKey platform** and not a production one! The JTAG/SWD is opened and the internal flash of the STM32 can be (re)programmed at will. Instructions about how to move from an open platform to a production one are summarized here: https://wookey-project.github.io/tataouine/production.html. Since we want this platform to remain open for future development purposes, **please do not lock it**. The production boards that will be provided later will have their MCU locked.

The JTAG/SWD connector can be found on the back side (Figure 2), and must be connected to a JTAG flasher (see below, this is a Discovery board in our case). Other noticeable elements on the back side of WooKey are the reset and DFU buttons, obviously used to reset the board and respectively put it in DFU mode when pressed at boot time.

(2) **Three smart card tokens** (see Figure 3) to flash the AUTH, DFU and SIG applets and play the associated role. These tokens are tagged in order to avoid confusing them and possibly lock them

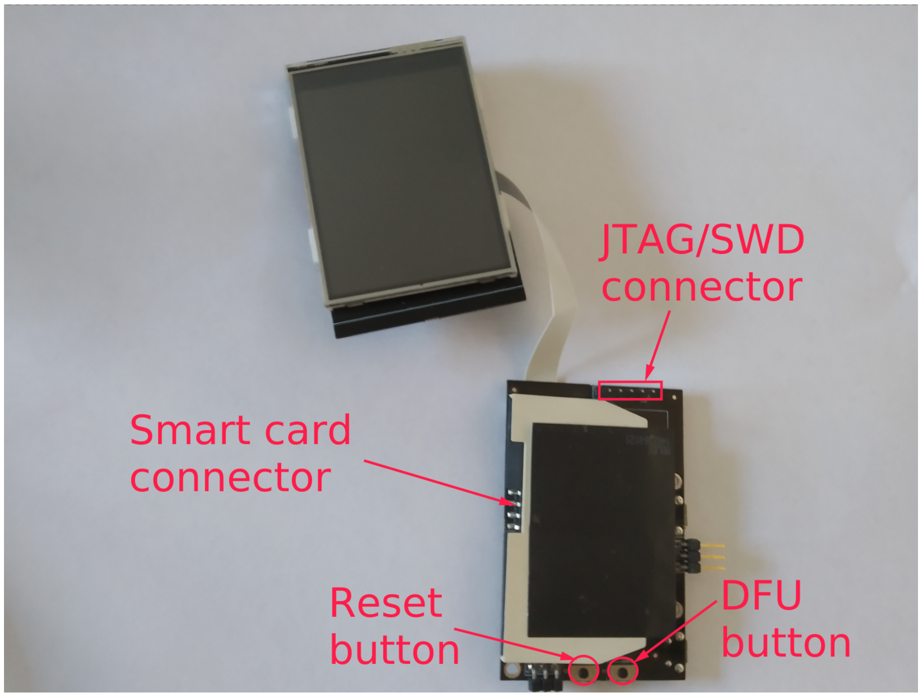

Fig. 2. WooKey main PCB (back), screen PCB and screen

inadvertently. These tokens are EAL4+ evaluated NXP JCOP J3D081 smart cards (see https://wookey-project.github.io/javacard/index.html). The tokens use the default GlobalPlatform secure channel key `0x404142434445464748494a4b4c4d4e4f` for (re)programming them: since we plan to use them for development purposes, **please do not modify them**. In a real production environment, the default key should be obviously changed after the provisioning phase in order to avoid attacks (see https://wookey-project.github.io/tataouine/production.html?highlight=globalplatform#lock-the-javacard-globalplatform-tokens). The production boards that will be provided later will have their tokens locked.

(3) **A STM32F429I-Discovery1 board** (see Figure 4) that plays the role of JTAG/SWD flasher in order to program the WooKey device internal flash (for the first time, before the DFU mode is available: in production JTAG/SWD should be obviously deactivated through RDP level 2 readout protection fuses).
The STM32F429G-Discovery1 board must be connected to the WooKey main board SWD/JTAG connector on one side, and to a host PC USB connector on the other side. Figure 5 shows how to perform this. Figure 6 details the connection mapping between the Discovery1 board JTAG/SWD five pins connector (numbered from 1 to 5) and WooKey's JTAG/SWD connector.
The STM32Discovery1 board should present itself as a ST-link device on the PC host once connected. JTAG flashing software such as `openocd` or `st-flash` can then be used.

Fig. 3. WooKey AUTH, DFU and SIG tokens

In order to get back the UART console from the WooKey board, the RX/TX PIN of the Discovery board (right next to the SWD connectors) can be connected directly to the USART1 RX/TX PIN of the WooKey. The serial line is then accessible through the Discovery serial console (usually /dev/ttyACM0). This method avoids the usage of a separated UART TTL adapter, as shown in Figure 7. When using the Discovery as UART/USB adapter, there is no need to connect the WooKey USART GND pin as the GND is distributed through the SWD link. If you wish to disconnect the SWD, link the USART GND to any GND GPIO on the Discovery board.

One last element will be necessary to program the tokens and is **not provided**, as it is expected to be widely and easily available:

(1) **A smart card reader** (see Figure 8) in order to flash the tokens with the AUTH/DFU/SIG applets, and in order to encrypt/sign firmwares that will be pushed through the DFU protocol to the WooKey in DFU mode.

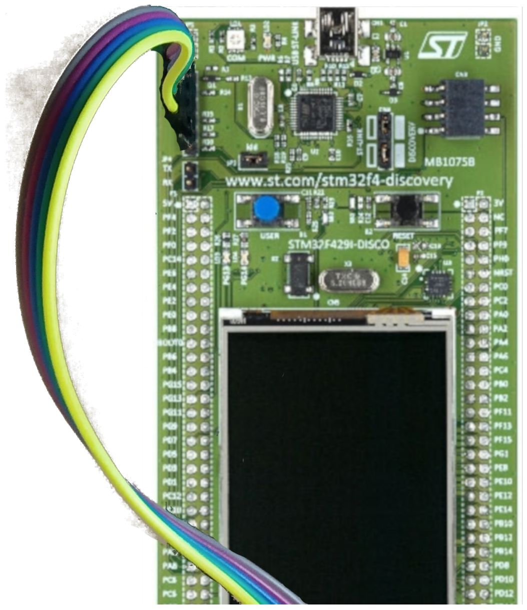

Fig. 4. STM32 Discovery board JTAG/SWD flasher

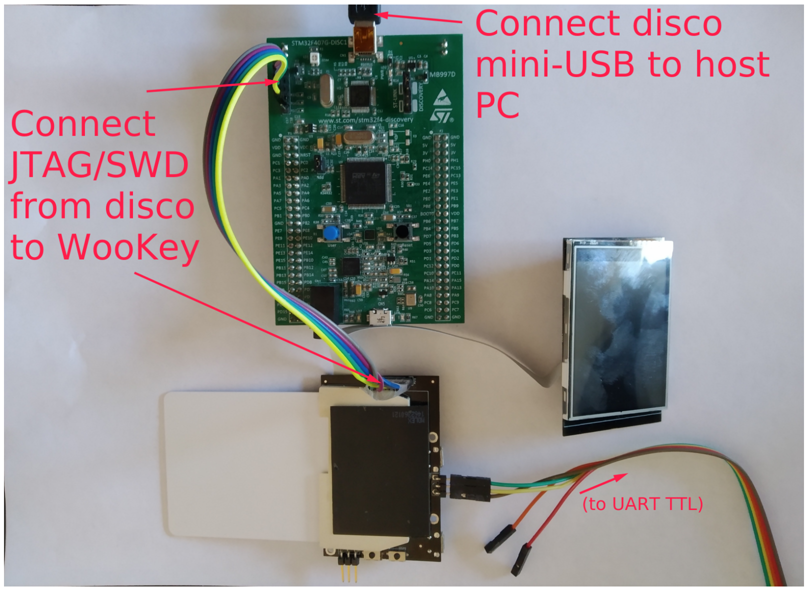
Connect disco mini-USB to host PC
Connect JTAG/SWD from disco to WooKey
(to UART TTL)

Fig. 5. STM32 Discovery board JTAG/SWD flasher: connection to WooKey and to PC host

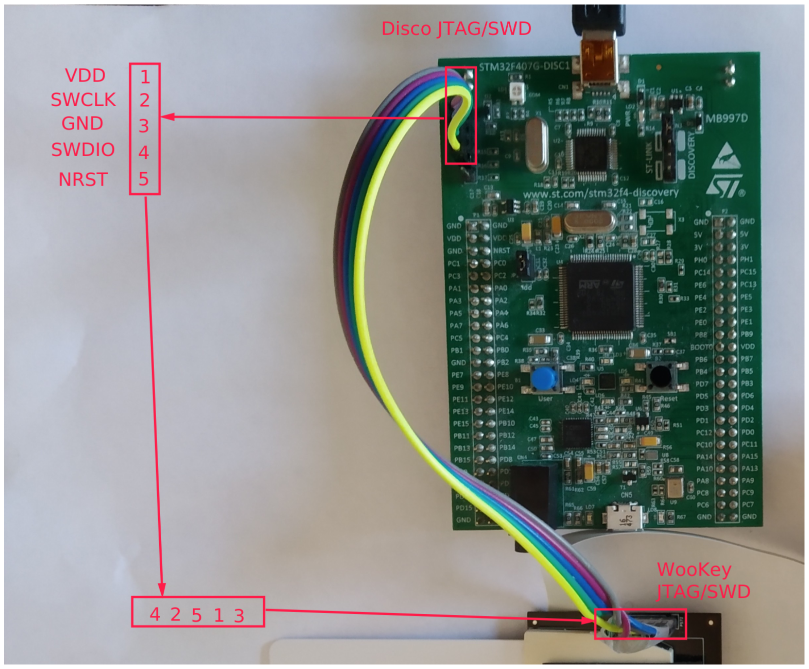

Fig. 6. STM32 Discovery board JTAG/SWD to WooKey JTAG/SWD mapping

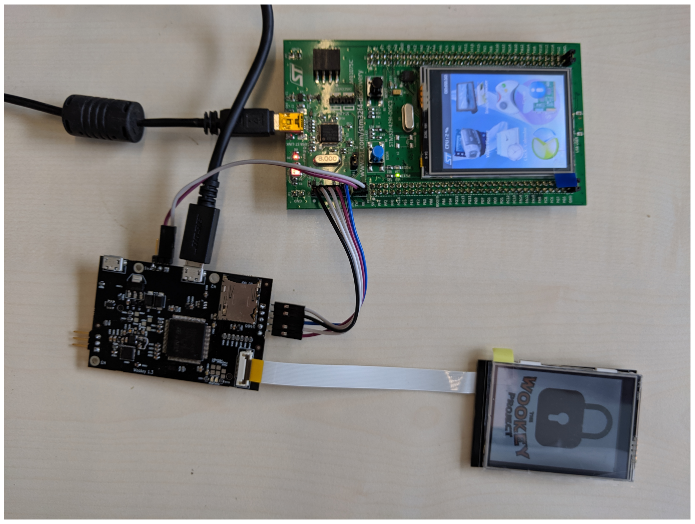

Fig. 7. Fully connected WooKey device, including UART

Your final setup should look like something similar to what is represented on Figure 7: the WooKey board is connected to the PC host through its USB HS or FS connector, it is connected to the discovery RX/TX using two wires (RX/TX), and it is connected to the Discovery flasher through a five wires JTAG/SWD

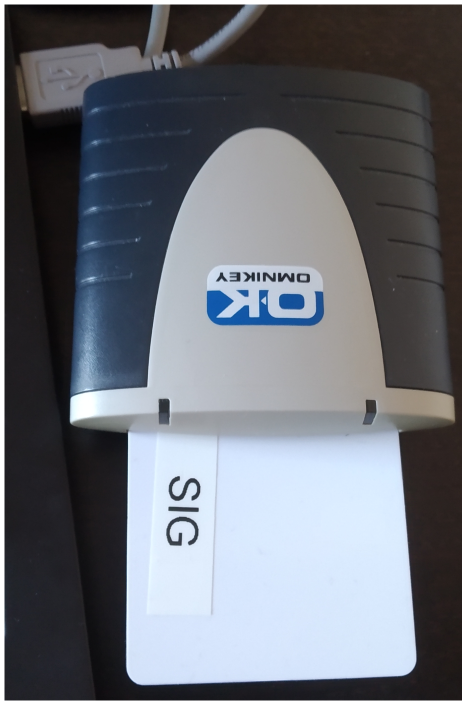

Fig. 8. Smart card reader

connector. The Discovery flasher is connected to a PC host USB port through its mini-USB connector. All in all, two USB ports should be used on your host PC in this setup. Beware that in order to properly flash the WooKey device, both the Discovery flasher and the WooKey board (through its USB HS or FS connector) **must** be connected to the host PC. Once the device is flashed, and in order to test the nominal and DFU modes, you can **disconnect** the Discovery flasher from the PC and the WooKey board, but leaving it connected is not an issue and makes it easier to flash it again. Of course, if you want to debug the running firmware through `gdb`, keeping the Discovery flasher connected is necessary.

NOTE: Please be aware that this (rather complex) setup is only here for debug/reviewing purposes and to show/debug the internals of the device. Of course, production WooKey boards **only use the USB HS or FS connector** as a regular USB mass storage would do!

NOTE: We provide a **fully flashed WooKey board** in the hardware delivery, as well as **programmed tokens**. You can directly test the nominal and DFU modes out of the box. If you decide to recompile your own firmware to test the compilation and flashing process, or to perform debugging, code modification and so on, please be aware that you will also have to compile the Javacard applets and program the tokens again in order to keep a **cryptographic consistency** for the whole system.

4 SOFTWARE DELIVERY: THE DOCKER FILE

In order to compile a firmware and flash a WooKey board, you can install all the dependencies and use the SDK as thoroughly detailed here: https://wookey-project.github.io/quickstart.html. However, in order

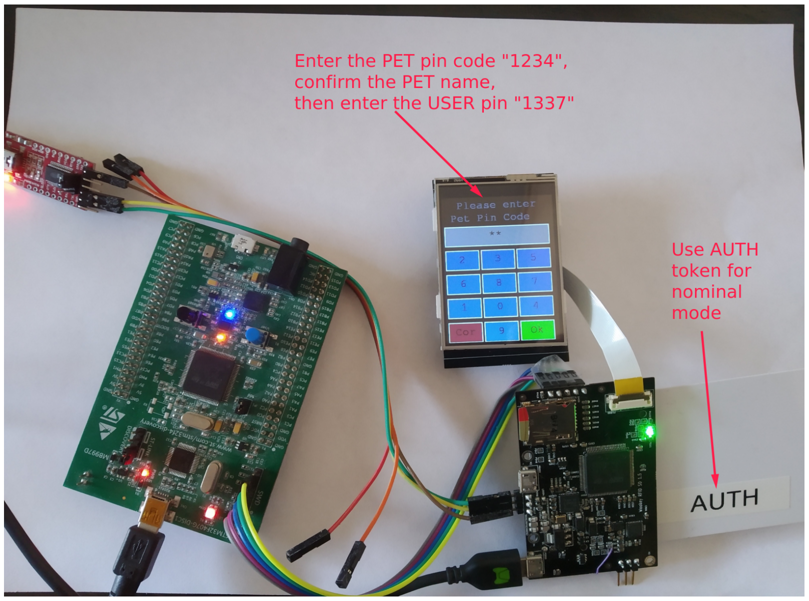

Fig. 9. Nominal mode overview

to make it easier and avoid distro/OS/dependencies issues, we provide a *Docker file* to build a dedicated container that embeds the WooKey project sources and all the necessary dependencies and tools to interact with a WooKey. However, be aware that since the Docker environment is **volatile**, you will have to save (if needed) all the data you want to keep, e.g. compiled firmware binaries, compiled applets, and more importantly the **private folder** containing all the cryptographic material that allows to keep a consistency between the WooKey platform and the AUTH/DFU/SIG tokens. Please follow the hints provided in the documentation at https://github.com/wookey-project/wookey-docker/blob/master/README.md in order to handle a persistent storage bound to your Docker host.

Hence, the only dependency needed here is obviously a *Linux distro running Docker*. The Docker build file can be found here: https://github.com/wookey-project/wookey-docker. Get it using git:

```
git clone https://github.com/wookey-project/wookey-docker
```

and follow the instructions of the README.md documentation. Concerning the configuration file, please choose `'boards/wookey/configs/wookey2_graphic_ada_fs_defconfig'` (for USB FS) or `'boards/wookey/configs/wookey2_graphic_ada_hs_defconfig'` (for USB HS) so that you will have debug output on the UART.

The firmware revision under evaluation corresponds to the manifest snapshot `wookey_20190903.xml` found here: https://github.com/wookey-project/manifest/blob/master/soft/snapshots/wookey_20190903.xml.

5 THE NOMINAL MODE OF WOOKEY

```
[211774.819610] usb 1-1.3: new high-speed USB device number 50 using xhci_hcd
[211774.916664] usb-storage 1-1.3:1.0: USB Mass Storage device detected
[211774.920917] scsi host2: usb-storage 1-1.3:1.0
[211775.937158] scsi 2:0:0:0: Direct-Access     ANSSI    wookey           0001 PQ: 0 ANSI: 0
[211775.938191] sd 2:0:0:0: Attached scsi generic sg1 type 0
[211775.939098] sd 2:0:0:0: [sdc] 7791616 4096-byte logical blocks: (31.9 GB/29.7 GiB)
[211775.939482] sd 2:0:0:0: [sdc] Write Protect is off
[211775.939491] sd 2:0:0:0: [sdc] Mode Sense: 03 00 00 00
[211775.940046] sd 2:0:0:0: [sdc] No Caching mode page found
[211775.940059] sd 2:0:0:0: [sdc] Assuming drive cache: write through
[211775.951880] sd 2:0:0:0: [sdc] Attached SCSI disk
```

Fig. 10. Linux kernel detecting WooKey after user authentication

```
$ sudo dd iflag=sync if=/dev/sdb of=/dev/null bs=32768 count=10000
10000+0 enregistrements lus
10000+0 enregistrements écrits
327680000 octets (328 MB, 312 MiB) copiés, 56,7693 s, 5,8 MB/s

$ sudo dd oflag=sync if=/dev/zero of=/dev/sdb bs=32768 count=1000
1000+0 enregistrements lus
1000+0 enregistrements écrits
32768000 octets (33 MB, 31 MiB) copiés, 7,23177 s, 4,5 MB/s
```

Fig. 11. Performing a dd test on the device

When you have compiled and flashed your firmware using the Discovery flasher, you can reset your WooKey board and you should be granted with the welcome screen with a lock on it (see Figure 7). WooKey is in **nominal mode**, and to be able to see it as a mass storage device on your host you must unlock it using the **AUTH token**.

Once the AUTH token is inserted, the green led should be switched on and you should be granted with a screen to enter your PET pin: enter *1234* using the touch screen. Then, you will have to confirm the PET name (a secret sentence that the user configures and stored in the token). Once confirmed, you can enter your USER pin *1234*. If everything is fine, WooKey launches its main unlocked screen, and a **USB mass storage device** should appear on the host. If the host is Linux, you should see the log on Figure 10 appear (the device `sdc` can of course vary). From now on, you should be able to handle your WooKey as any mass storage device, by performing raw `dd` on it (see Figure 11), creating partitions using `fdisk`, creating a file system using `mkfs`, and so on. A blue led should now indicate SCSI activity on the device.

NOTE: You should also see EwoK/WooKey tasks logs on your UART minicom console.

NOTE: You can use a sharp object to enter elements on the touch screen, for instance the edge of a smart card. This usually yields in better precision.

NOTE: It is possible to debug the firmware on WooKey using `gdb` through `openocd`, see here: https://wookey-project.github.io/quickstart.html#debug-with-gdb. Please be aware that debugging will halt the device, and as a consequence will break connectivity with the host.

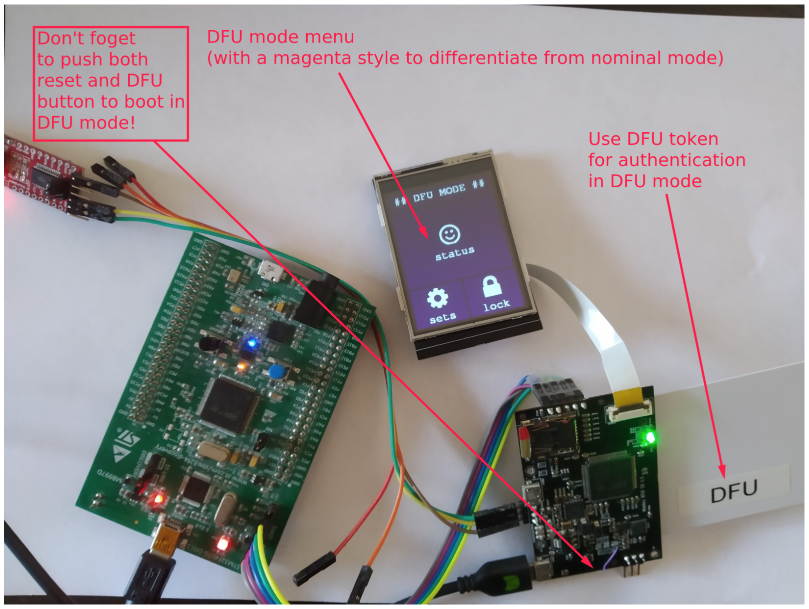
Don't foget to push both reset and DFU button to boot in DFU mode!
DFU mode menu (with a magenta style to differentiate from nominal mode)
Use DFU token for authentication in DFU mode
DFU MODE
status
sets
lock
DFU

Fig. 12. Entering the DFU mode

6 THE DFU MODE OF WOOKEY

In order to enter the DFU mode of WooKey, you must reboot it **while holding the DFU button** for a few seconds. You will have to use the dedicated **DFU token** in order to authenticate and boot the platform. You should then see a magenta style menu indicating that you have indeed entered the DFU mode. Enter you PET pin *1234*, confirm PET name, and enter USER pin *1234*[1].

Now the DFU mode unlocked menu should appear as presented on Figure 12, and a DFU device with `dead:cafe` USB ID should appear on your host. From now on you can use the `dfu-util` tool to upload a **signed and encrypted firmware** to the device as shown on Figure 13 (the blue led should also show activity while processing the firmware chunks). Before accepting to download a new firmware, the device asks the user a confirmation of the version of the firmware (see Figure 14), and when accepted a DFU in progress dialog is shown (see Figure 15). When the firmware has been successfully downloaded and cryptographically verified by WooKey, a reset is performed to boot on the new firmware.

On the host side, a firmware can be encrypted and signed using the SIG token inserted in the smart card reader plugged using the commands on Figure 16 (`pcscd` is the daemon used to drive the smart card reader). The user is asked for the SIG PET pin *1234*, to confirm the SIG PET name, and asked for the SIG USER pin *1234*. Please note the `'tosign=flip:flop'` argument asking to sign both flip and flop firmwares, as well as the `'version="1.0.0-0"'` asking to set this specific version. Be aware that WooKey has a **strict anti-rollback** policy, so you will have to specify a version greater that the one in the device. Also, if the device is running in flip mode, you must send a flop update (and vice versa). The produced files should be in `build/armv7-m/wookey/flip_fw.bin.signed` and `build/armv7-m/wookey/flop_fw.bin.signed`.

[1]Please note that for DFU mode, the default USER pin is *1234* and differs from the one in nominal mode.

```
$ sudo /usr/bin/dfu-util -v -D build/armv7-m/wookey/flop_fw.bin.signed -t 4096 -d dead:cafe
dfu-util 0.9

Copyright 2005-2009 Weston Schmidt, Harald Welte and OpenMoko Inc.
Copyright 2010-2016 Tormod Volden and Stefan Schmidt
This program is Free Software and has ABSOLUTELY NO WARRANTY
Please report bugs to http://sourceforge.net/p/dfu-util/tickets/

DFU suffix version 100
Opening DFU capable USB device...
ID dead:cafe
Run-time device DFU version 0110
Claiming USB DFU Interface...
Setting Alternate Setting #0 ...
Determining device status: state = dfuIDLE, status = 0
dfuIDLE, continuing
DFU mode device DFU version 0110
Copying data from PC to DFU device
Download [=========================] 100%       933888 bytes
Download done.
Sent a total of 933888 bytes
state(2) = dfuIDLE, status(0) = No error condition is present
Done!
```

Fig. 13. Performing a `dfu-util` to upload a signed firmware

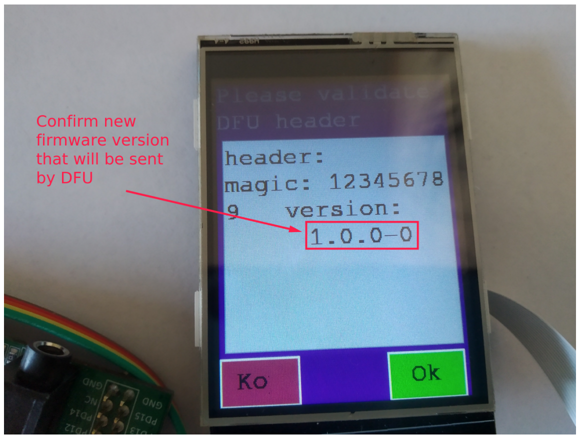
Confirm new firmware version that will be sent by DFU
Please validate DFU header
header:
magic: 12345678
9 version:
1.0.0-0
Ko
Ok

Fig. 14. Firmware version check and validation during DFU mode

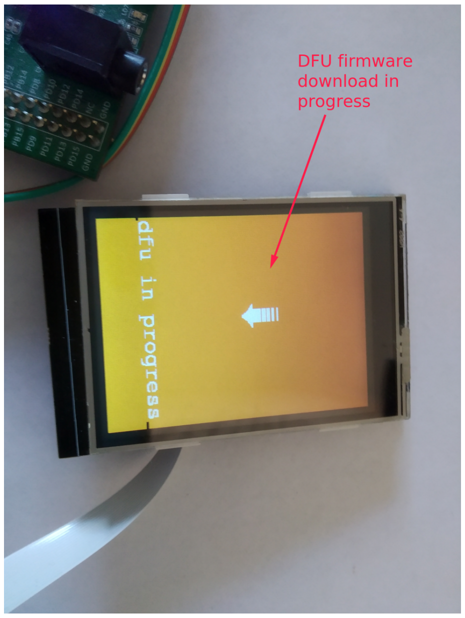

Fig. 15. DFU mode firmware download progress

```
$ sudo /usr/sbin/pcscd
$ make sign_interactive tosign=flip:flop version="1.0.0-0"
```

Fig. 16. Signing and encrypting a firmware on the host side using the SIG token