Is the Pen mightier than the Sword?

A First Look Into The Security of The Apple Pencil and the Apple Smart Keyboard

August, 2018

Who am I?

Stefan Esser

- working in IT Security since 1998
- started with runtime encryption / decryption
- moved on to linux daemon security
- then did a lot of work in PHP and Web Application Security
- in 2010 moved into the field of iOS security

Motivation

- Apple Pencil and Apple Smart Keyboard are popular accessories for iPads
- lot of people analyse the security of iOS itself
- but no public research on security of these accessories
- in theory these devices can be easily replaced by malicious ones without user noticing
- users might even borrow these devices to their seat neighbour in a coffee shop.
- the best place for a key logger is likely in the firmware of the keyboard
- can a malicious accessory take over the iPad?

Disclaimer

- this is meant as the first introductory talk on this topic
- we want to get the research in these accessory started
- more talks will follow
- so far no devices were harmed during this research - that means no hardware attack, yet

Open Source ?!?

- Resources for this talk will be shared on:
  - https://github.com/Antid0teCom/ipad_accessory_research

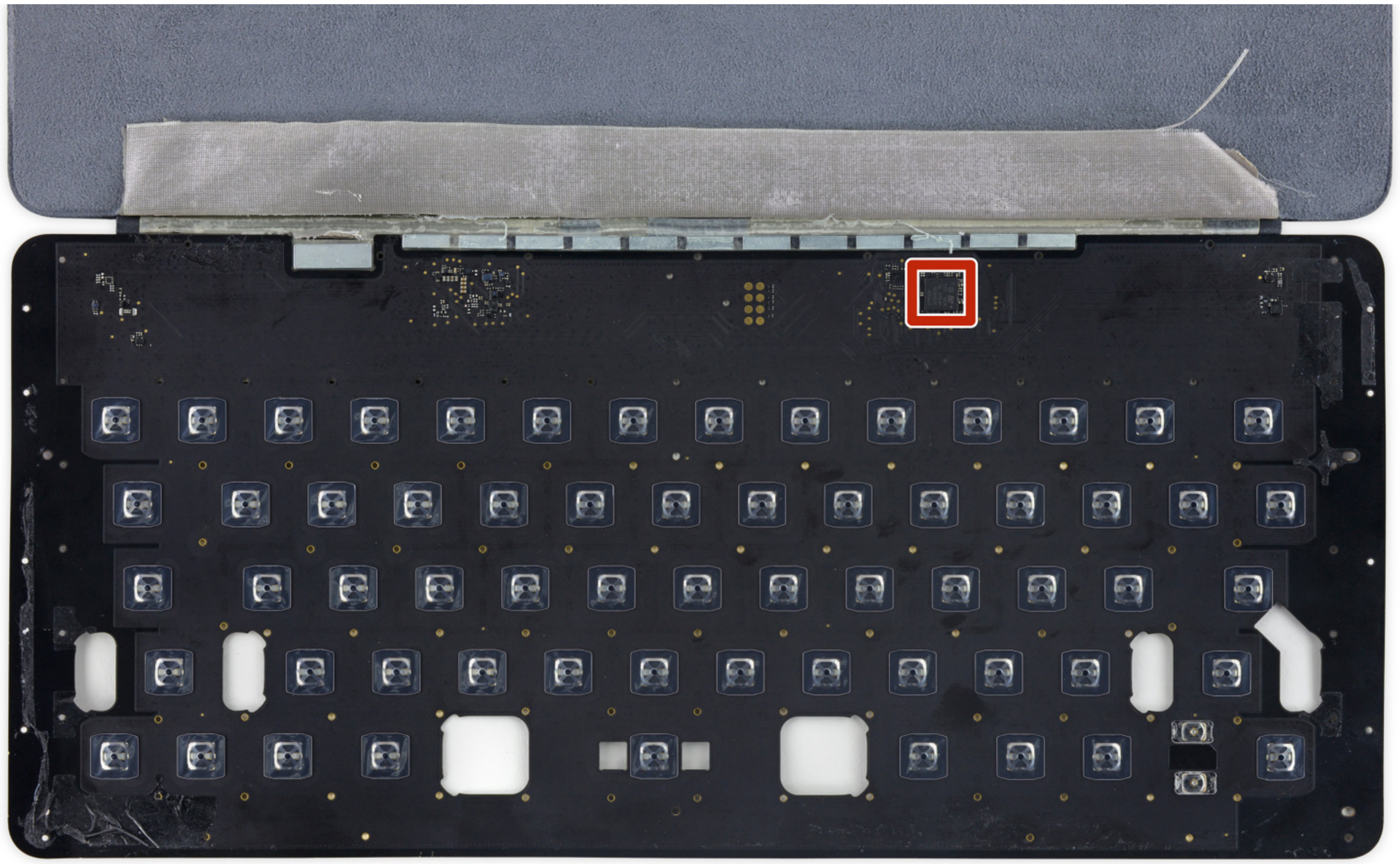
- Few things are already available - other stuff will be uploaded during the next days

Accessory Teardowns

Apple Smart Keyboard

- connected via the Smart Connector
- three different form factors
  - 12.9“ - A1636
  - 10.5“ - A1829
  - 9.7“ - A1772
- IFIXIT teardown of A1636 says it uses:

  STM32F103VB microcontroller



source: https://www.ifixit.com/Teardown/Smart+Keyboard+Teardown/53052
https://www.ifixit.com/Guide/Image/meta/ZfFJCjwWC1DKCQTQ
photo by: Sam Lionheart
license: Creative Commons BY-NC-SA

ATTENTION: for A1772 we assume the same microcontroller is used - for A1829 no firmware/teardown is available

STM32F103VB

- Product URL: https://www.st.com/en/microcontrollers/stm32f103vb.html
- Datasheet: https://www.st.com/resource/en/datasheet/stm32f103vb.pdf

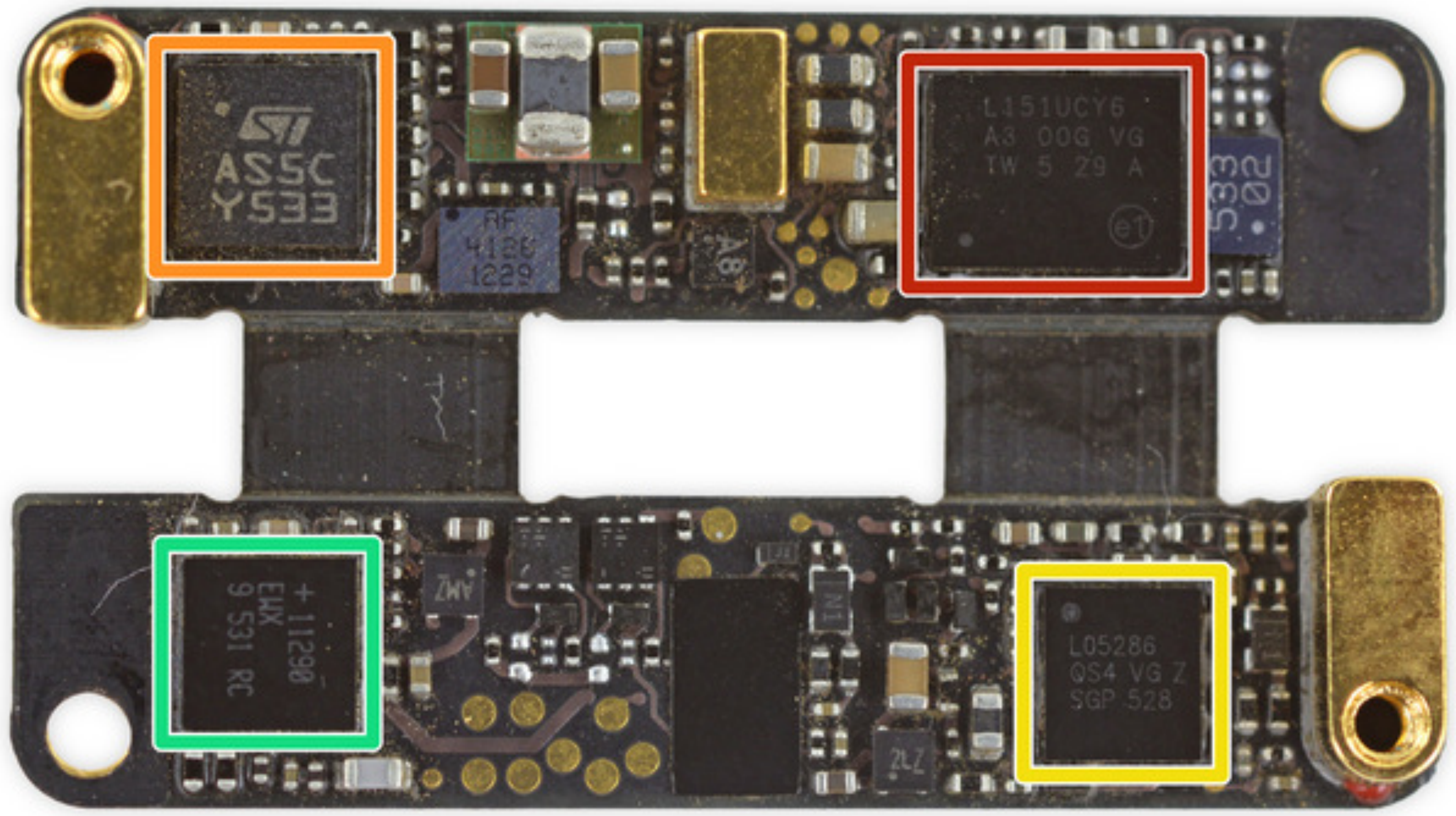
- Mainstream Performance line
- ARM Cortex-M3 MCU
- 128 Kbytes Flash
- 20 Kbytes SRAM
- Memory Protection Unit: **NO**

Apple Pencil

- connected via Lightning / Bluetooth

- IFIXIT teardown says it uses:

  STML151UCY6 microcontroller



source: https://www.ifixit.com/Teardown/Apple+Pencil+Teardown/52955
https://www.ifixit.com/Guide/Image/meta/5ibhjx6plRvgpqfU
photo by: Sam Lionheart
license: Creative Commons BY-NC-SA

STM151UCY6

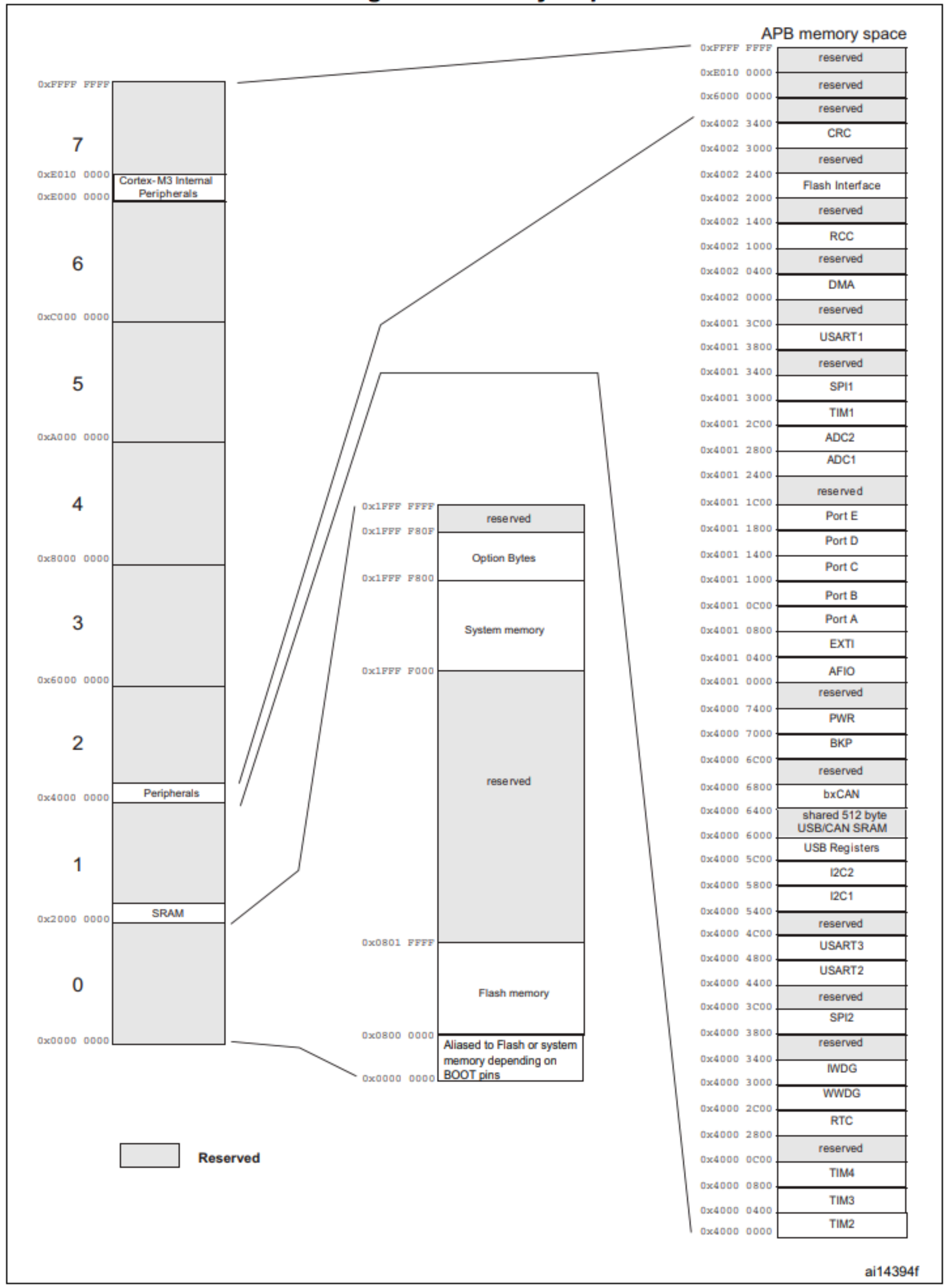
- Product URL: https://www.st.com/en/microcontrollers/stm32l151cc.html
- Datasheet: https://www.st.com/resource/en/datasheet/stm32l151cc.pdf

- Ultra-low-power
- ARM Cortex-M3 MCU
- 256 Kbytes Flash
- 32 Kbytes SRAM
- 8 Kbytes Data EEPROM
- Memory Protection Unit: **YES**
- Memory Option Bytes: for memory read-out protection

STM Microcontrollers

- very well documented
- product URL leads to a website with full documentation
- even programming guides
- everything one needs to know to understand low level programming

Figure 11. Memory map



Memory Protection Unit (MPU)

- allows to have memory with different protection levels R / W / X

- Apple Smart Keyboard doesn't have it according to data sheet
  - no protection on memory

- Apple Pencil has support for MPU but we did not see the firmware configuring it
  - no protection on memory (unless unknown bootrom configures MPU)

Accessory Firmware Updates

Accessory Firmware

- both Apple Pencil and Apple Smart Keyboard allow firmware upgrades
- firmware is automatically upgraded when connected to iPad
- upgrade seems to be silent without user notification
- user not involved in OTA update process
- to get copy of firmware we need to follow the upgrade discovery path

Accessory Firmware - Who?

- Launch Daemon:
  - com.apple.MobileAccessoryUpdater

- starts **fud** daemon:
  - `/System/Library/PrivateFrameworks/MobileAccessoryUpdater.framework/Support/fud`
  - runs as root
  - **without a sandbox (!!!)**

Accessory Firmware - Who? (II)

- the **fud** daemon uses the private **MobileAccessoryUpdater.framework**
- it will load plugin bundles from `/System/Library/AccessoryUpdaterBundles/`
  - `AppleEAAccessoryUpdater.bundle`
  - `AppleEmbeddedAccessoryUpdater.bundle`
  - `FUDGenericKextUpdater.bundle`
  - `StandaloneHIDFudPlugins.bundle`
- **StandaloneHIDFudPlugins** is responsible for Apple Pencil / Apple Smart Keyboard
- by reversing this plugin we can learn the HIDDevice connections are used for the upgrade

Accessory Firmware Upgrade Discovery (I)

- in order to discover firmware upgrades a firmware upgrade manifest must be downloaded

  - Apple Pencil
    https://mesu.apple.com/assets/com_apple_MobileAsset_MobileAccessoryUpdate_WirelessStylusFirmware/com_apple_MobileAsset_MobileAccessoryUpdate_WirelessStylusFirmware.xml

  - Apple Smart Keyboard 12.9“
    https://mesu.apple.com/assets/com_apple_MobileAsset_MobileAccessoryUpdate_KeyboardCoverFirmware/com_apple_MobileAsset_MobileAccessoryUpdate_KeyboardCoverFirmware.xml

Accessory Firmware Upgrade Discovery (II)

- Reversing revealed the following manifest URLs that are currently either empty or access is denied - **NO FIRMWARE BINARY AVAILABLE**

  - Apple Smart Keyboard 9.7”
    https://mesu.apple.com/assets/com_apple_MobileAsset_MobileAccessoryUpdate_MiniKeyboardCoverFirmware/com_apple_MobileAsset_MobileAccessoryUpdate_MiniKeyboardCoverFirmware.xml

  - Apple Smart Keyboard 10.5“
    https://mesu.apple.com/assets/com_apple_MobileAsset_MobileAccessoryUpdate_KeyboardCoverFirmware_3/com_apple_MobileAsset_MobileAccessoryUpdate_KeyboardCoverFirmware_3.xml

Accessory Firmware Upgrade Manifest (I)

- AFU Manifest in XML plist format
- defines what version is available
- and how it is packaged
- usually in ZIP format
- …

```xml
<?xml version="1.0"?>
<plist version="1.0">
  <dict>
    <key>AssetType</key>
    <string>com.apple.MobileAsset.MobileAccessoryUpdate.WirelessStylusFirmware</string>
    <key>Assets</key>
    <array>
      <dict>
        <key>FirmwareVersions</key>
        <dict>
          <key>14</key>
          <dict>
            <key>1</key>
            <integer>584</integer>
          </dict>
          <key>15</key>
          <dict>
            <key>1</key>
            <integer>584</integer>
          </dict>
        </dict>
        <key>_CompatibilityVersion</key>
        <integer>2</integer>
        <key>_CompressionAlgorithm</key>
        <string>zip</string>
        <key>_ContentVersion</key>
        <integer>248</integer>
        <key>_DownloadSize</key>
        <integer>244355</integer>
        <key>_IsZipStreamable</key>
        <true/>
        …
```


Accessory Firmware Upgrade Manifest (I)

- ...
- contains download URL
- SHA-1 hash of firmware file
- manifest is signed with AssetManifestSigning certificate


```xml
<key>_MasteredVersion</key>
        <string>1611</string>
        <key>_Measurement</key>
        <data> tRitODC6Kg8+IhT+VA5Rh5celjE= </data>
        <key>_MeasurementAlgorithm</key>
        <string>SHA-1</string>
        <key>_UnarchivedSize</key>
        <integer>387072</integer>
        <key>__BaseURL</key>
        <string>http://appldnld.apple.com/ios10.0/.../</string>
        <key>__CanUseLocalCacheServer</key>
        <true/>
        <key>__InstallWithOS</key>
        <true/>
        <key>__RelativePath</key>
        <string>com_..._MobileAccessoryUpdate_WirelessStylusFirmware/xxx.zip</string>
      </dict>
    </array>
    <key>Certificate</key>
    <data>...</data>
    <key>FormatVersion</key>
    <integer>1</integer>
    <key>Signature</key>
    <data>...</data>
    <key>SigningKey</key>
    <string>AssetManifestSigning</string>
  </dict>
</plist>
```

Accessory Firmware Upgrade Manifest Certificate

- AssetManifestSigning certificate **expired in July 2018**
- impact of expiry on firmware signing process is unknown
- firmware seems to continue to upgrade on existing devices

- **UNANSWERED QUESTION:** does it stop firmware upgrades on devices restored after that date?

```
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number: 45 (0x2d)
    Signature Algorithm: sha1WithRSAEncryption
        Issuer: C=US, O=Apple Inc., OU=Apple Certification Authority, CN=Apple iPhone Certification Authority
        Validity
            Not Before: Jul 14 22:32:48 2011 GMT
            Not After : Jul 14 22:32:48 2018 GMT
        Subject: C=US, O=Apple Inc., OU=Apple iOS Asset Manifest, CN=Asset Manifest Signing
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                Public-Key: (2048 bit)
                Modulus:
                    00:b5:cb:80:f8:55:80:4f:1c:bc:27:1e:4d:6a:7e:
                    2d:45:e3:d1:37:32:5e:bb:e4:49:58:60:01:60:86:
                    aa:40:88:a3:aa:79:a8:0b:55:95:3c:cc:ab:8f:c3:
                    ae:74:34:70:02:bc:d2:5b:8c:a0:63:32:c5:9d:59:
                    a1:4c:8f:fe:dc:d9:30:79:22:42:70:8b:ad:58:e8:
                    1f:ae:54:ae:fc:5b:db:bd:23:f8:45:00:ad:29:59:
                    c3:3d:63:92:9b:28:cb:f3:e3:01:30:b7:ae:04:5d:
                    f4:bc:79:50:51:9a:a8:7f:db:dc:a0:c4:df:4d:b4:
                    16:c7:12:21:a2:19:0f:2f:c4:85:77:53:a1:68:98:
                    d7:66:c4:a3:cc:ed:56:66:b3:21:48:c5:0e:47:b3:
                    18:07:6f:4b:24:c6:50:c8:75:e3:ed:62:c1:cb:9a:
                    92:bd:3d:7e:37:2b:7b:01:4f:79:47:37:45:31:b6:
                    2b:7c:1d:3a:dd:c2:23:6a:d7:77:08:d1:32:0d:4f:
                    e9:6c:6d:72:8b:a7:7f:e0:3c:95:69:7f:19:10:dc:
                    c4:ed:e3:42:86:fc:34:cc:b2:a2:8e:ca:00:e8:99:
                    bb:05:4a:a0:3e:44:f4:af:eb:c6:2d:27:f9:00:68:
                    66:a8:1f:a8:19:7d:1a:f0:5f:b1:89:a3:3c:d0:f0:
                    a4:d9
                Exponent: 65537 (0x10001)
        X509v3 extensions:
            X509v3 Key Usage: critical
                Digital Signature
            X509v3 Basic Constraints:
                CA:FALSE
            X509v3 Subject Key Identifier:
                63:50:C4:FE:B2:D4:0A:38:1E:B8:62:77:A0:5C:30:BC:1C:AC:1D:C1
            X509v3 CRL Distribution Points:
                 ...
```

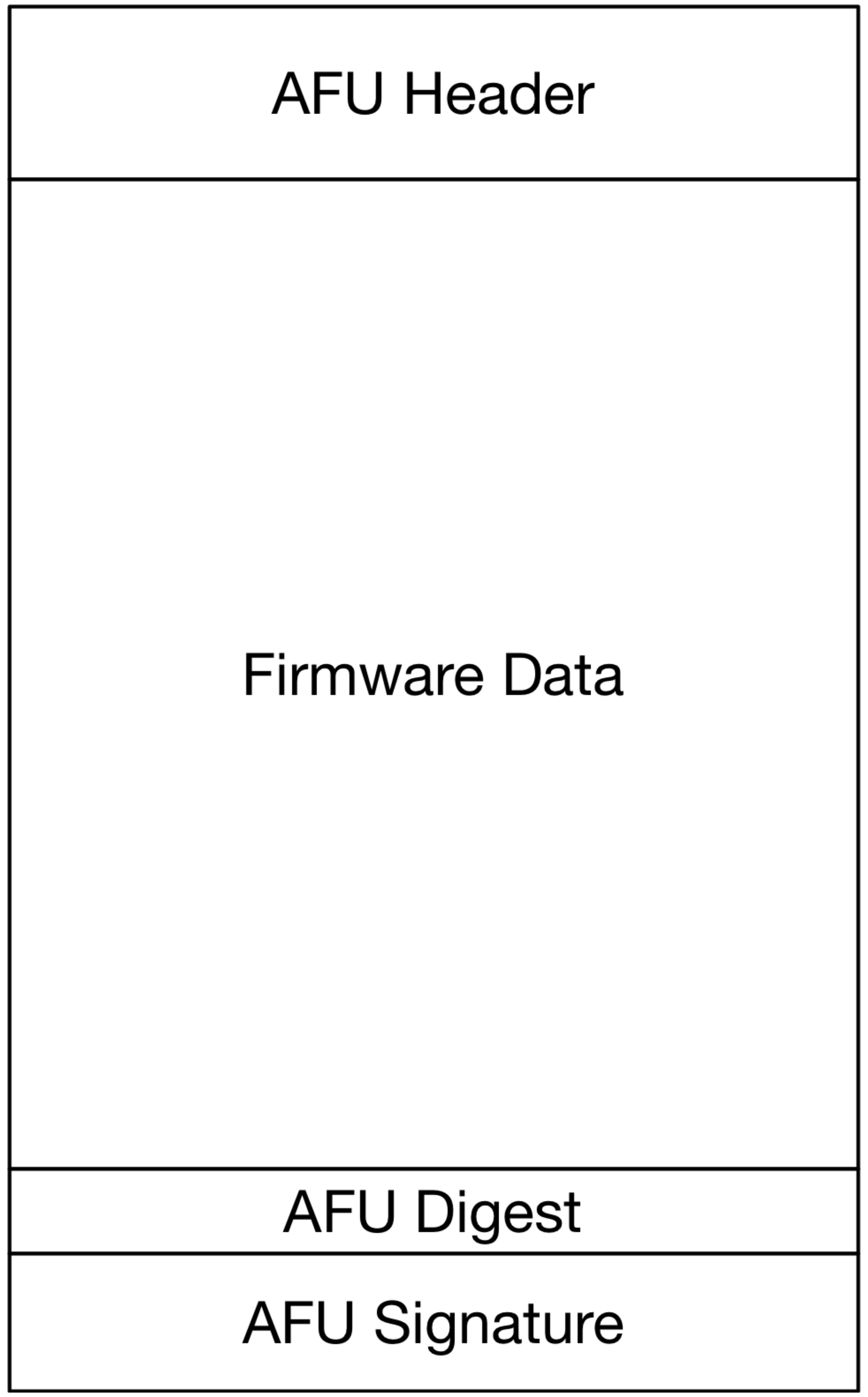

Accessory Firmware

Accessory Firmware

- accessory firmware for Apple Pencil and Smart Keyboard comes in AFU files
- file extension is not necessarily .afu
- other accessories like Smart Remote use it too
- but e.g. AirPods use a different format

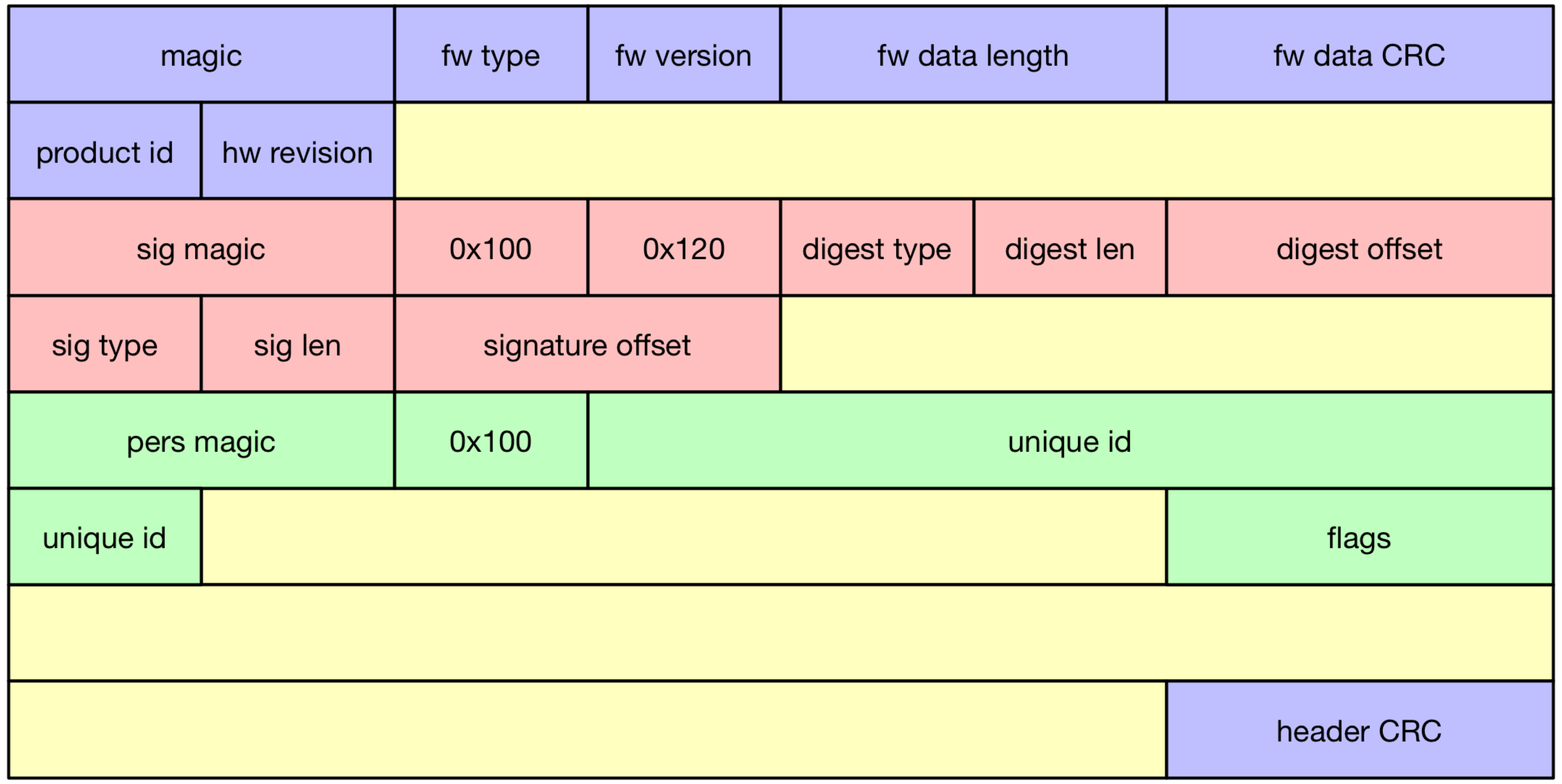
AFU File Format

- AFU files have a 128 byte header
- followed by the firmware data
- followed by a SHA256 hash of header plus data
- at the end there is a RSA2048 signature



AFU File Header

- main header
  - firmware type, firmware version
  - product / hw revision
  - data length and CRC
  - CRC algorithm is either standard CRC32 or STM32 depending on accessory type
- signature header
  - theoretically optional
  - defines and locates digest and RSA signature
- personalization header
  - optional
  - hardcodes accessory unique id



AFU File Tools

- AFU_check.py
  - python script that verifies the CRCs, the digest and the RSA signature of an AFU file

- AFU_loader.py
  - IDAPython File Format Loader Plugin that adds AFU file format support to IDA Pro

Apple Pencil Firmware

- current version: 2.48
- available as OTA Update
- 183kb in size
- firmware data loaded at 0x8006080
- around 1200 functions
- around 210 strings

Apple Smart Keyboard Firmware

- different form factors have different current firmware:
  - `12.9" - 0x855`
  - `9.7"  - 0x858`
  - `10.5" - 0x12d`
- OTA Update only available for 12,9" and only old firmware 0x850
- 45.8kb in size
- firmware data loaded at 0x08002600
- around 590 functions
- 6 strings

Accessory Bootloader

- firmware of Pencil and Smart Keyboard is not loaded at start of memory map
- it is unknown what is loaded before the firmware
- likely a small boot loader that cannot be upgraded by AFU
- so far it has not been extracted (, yet)

Apple Pencil

Apple Pencil

- Firmware for Apple Pencil has way more strings
- allows to understand parts a bit easier
- contains a RTXC by unknown vendor
- heap implementation with forward and backward coalescence
- has named tasks which again makes understanding easier

Apple Pencil Task List

- PowerMgr
- radio_interface
- idbus
- idbus_ea_radioDiags
- Force
- Inertia
- battery
- iap_idbus
- BTSync
- FWU_Deferred

```
__TEXT:08032284 ; struct _task_list_entry list_of_tasks_maybe
__TEXT:08032284 _list_of_tasks_maybe _task_list_entry <_powermgr_main+1, aPowermgr, 0x44C, 0x28, 1>
__TEXT:08032284                                         ; DATA XREF: sub_80203A0+6A↑o
__TEXT:08032284                                         ; sub_80203A0+92↑o ...
__TEXT:08032284                                         ; "PowerMgr"
__TEXT:08032298                 _task_list_entry <_radio_interface_main+1, aRadioInterface, 0x4B0, 0, \ ; "radio_interface"
__TEXT:08032298                                   0>
__TEXT:080322AC                 _task_list_entry <_idbus_main+1, aIdbus, 0x6A4, 0, 0> ; "idbus"
__TEXT:080322C0                 _task_list_entry <_idbus_ea_radioDiags_main+1, aIdbusEaRadiodi, 0x258,\ ; "idbus_ea_radioDiags"
__TEXT:080322C0                                   0, 0>
__TEXT:080322D4                 _task_list_entry <_Force_main+1, aForce, 0x4B0, 0, 0> ; "Force"
__TEXT:080322E8                 _task_list_entry <_Inertia_main+1, aInertia, 0x708, 0, 0> ; "Inertia"
__TEXT:080322FC                 _task_list_entry <_battery_main+1, aBattery, 0x3DE, 0, 0> ; "battery"
__TEXT:08032310                 _task_list_entry <_iap_idbus_main+1, aIapIdbus, 0x370, 0, 0> ; "iap_idbus"
__TEXT:08032324                 _task_list_entry <_BTSync_main+1, aBtsync, 0x400, 0, 0> ; "BTSync"
__TEXT:08032338                 _task_list_entry <_fwu_deferred_main+1, aFwuDeferred, 0x1800, 0x3C, 1> ; "FWU_Deferred"
__TEXT:0803234C
```

 |

Apple Pencil

- we already know that firmware upgrade uses HID so we need to find HID parsers
- command line tool **ioreg** reveals 4 HID devices with different primary usage

```
+-o Apple Pencil  <class IOAccessoryIDBusBulkData>
  +-o IOAccessoryIDBusBulkDataEndpoint@0  <class IOAccessoryIDBusBulkDataEndpoint>
  | +-o IOAccessoryIDBusHIDDevice  <class IOAccessoryIDBusHIDDevice>
  |   +-o IOHIDInterface  <class IOHIDInterface>
  +-o IOAccessoryIDBusBulkDataEndpoint@1  <class IOAccessoryIDBusBulkDataEndpoint>
  | +-o IOAccessoryIDBusHIDDevice  <class IOAccessoryIDBusHIDDevice>
  |   +-o IOHIDInterface  <class IOHIDInterface>
  +-o IOAccessoryIDBusBulkDataEndpoint@2  <class IOAccessoryIDBusBulkDataEndpoint>
  | +-o IOAccessoryIDBusHIDDevice  <class IOAccessoryIDBusHIDDevice>
  |   +-o IOHIDInterface  <class IOHIDInterface>
  +-o IOAccessoryIDBusBulkDataEndpoint@3  <class IOAccessoryIDBusBulkDataEndpoint>
  | +-o IOAccessoryIDBusHIDDevice  <class IOAccessoryIDBusHIDDevice>
  |   +-o IOHIDInterface  <class IOHIDInterface>
  +-o IOAccessoryIDBusBulkDataEndpoint@4  <class IOAccessoryIDBusBulkDataEndpoint>
    +-o IOAccessoryPortIDBus@6  <class IOAccessoryPortIDBus>
      +-o IOAccessoryPortUserClient  <class IOAccessoryPortUserClient>
```


Apple Pencil

- the HID Devices have report descriptors

"ReportDescriptor" =
<0600ff0a**0b**00a1010a0b0085ff7680069501b20201c00000000000000000000000000000000000000000000000000000000000000000000000000>
"ReportDescriptor" =
<0600ff0a**0e**00a1010a0e0085ff7680069501b20201c00000000000000000000000000000000000000000000000000000000000000000000000000>
"ReportDescriptor" =
<0600ff0a**11**00a1010a110085ff7680069501b20201c00000000000000000000000000000000000000000000000000000000000000000000000000>
"ReportDescriptor" =
<0600ff0a**12**00a1010a120085ff7680069501b20201c00000000000000000000000000000000000000000000000000000000000000000000000000>

- marked in red is the encoded primary usage: 11, 14, 17, 18
-

```
__TEXT:0802F778 _device_240     DCD 0xAFF0006, 0x1A1000B, 0x85000B0A, 0x68076FF, 0x2B20195
__TEXT:0802F778                                         ; DATA XREF: __TEXT:0802F68C↑o
__TEXT:0802F778                                         ; __TEXT:0802F704↑o
__TEXT:0802F778                 DCD 0xC001, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
__TEXT:0802F7B8 _device_243     DCD 0xAFF0006, 0x1A1000E, 0x85000E0A, 0x68076FF, 0x2B20195
__TEXT:0802F7B8                                         ; DATA XREF: __TEXT:0802F6A4↑o
__TEXT:0802F7B8                                         ; __TEXT:0802F71C↑o
__TEXT:0802F7B8                 DCD 0xC001, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
__TEXT:0802F7F8 _device_248     DCD 0xAFF0006, 0x1A10011, 0x8500110A, 0x68076FF, 0x2B20195
__TEXT:0802F7F8                                         ; DATA XREF: __TEXT:0802F6BC↑o
__TEXT:0802F7F8                                         ; __TEXT:0802F734↑o
__TEXT:0802F7F8                 DCD 0xC001, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
__TEXT:0802F838 _device_245     DCD 0xAFF0006, 0x1A10012, 0x8500120A, 0x68076FF, 0x2B20195
__TEXT:0802F838                                         ; DATA XREF: __TEXT:0802F6D4↑o
__TEXT:0802F838                                         ; __TEXT:0802F74C↑o
__TEXT:0802F838                 DCD 0xC001, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
__TEXT:0802F878 aComAppleDevdev DCB "com.apple.devDevice.radioDiags",0
__TEXT:0802F878                                         ; DATA XREF: __TEXT:0802F764↑o
__TEXT:0802F897                 DCB 0
```

Apple Pencil - Sub Devices

- from the HID report descriptors we can distinguish internal sub devices
- from tasks registering callbacks we get idea what sub device it is
  - primary usage 11 - code 240 - main
  - primary usage 14 - code 243 - inertia
  - primary usage 17 - code 248 - force
  - primary usage 18 - code 245 - radio

- Inertia_main:

```
set_hid_feature_report_callback(243,
device_243_get_feature_report_INERTIA,    <— parser for HID Get Feature Report
device_243_set_feature_report_INERTIA);   <— parser for HID Set Feature Report
```


AFU Firmware Upgrade Protocol

AFU Firmware Upgrade Protocol

- the iOS Device opens a connection to the accessory’s HID Device with primary usage 11
- **this can be done even from a sandboxed iOS process**

```
    CFMutableDictionaryRef matching_dict = IOServiceMatching("IOAccessoryIDBusHIDDevice");
    int32_t primary_usage = 11;
    int32_t product_id = 0x268;
    CFDictionaryAddValue(matching_dict, CFSTR("PrimaryUsage"), CFNumberCreate(kCFAllocatorDefault, kCFNumberSInt32Type, &primary_usage));
    CFDictionaryAddValue(matching_dict, CFSTR("ProductID"), CFNumberCreate(kCFAllocatorDefault, kCFNumberSInt32Type, &product_id));

    kr = IOServiceGetMatchingServices(0, matching_dict, &iter);

    service = IOIteratorNext(iter);
    while (service) {

        hiddevice = IOHIDDeviceCreate(kCFAllocatorDefault, service);

        if (hiddevice) {
            fprintf(f, "HID Device Created\n");
            kr = IOHIDDeviceOpen(hiddevice, 0);
            if (kr == 0) {
                fprintf(f, "HID Device Opened\n");
                ...
```

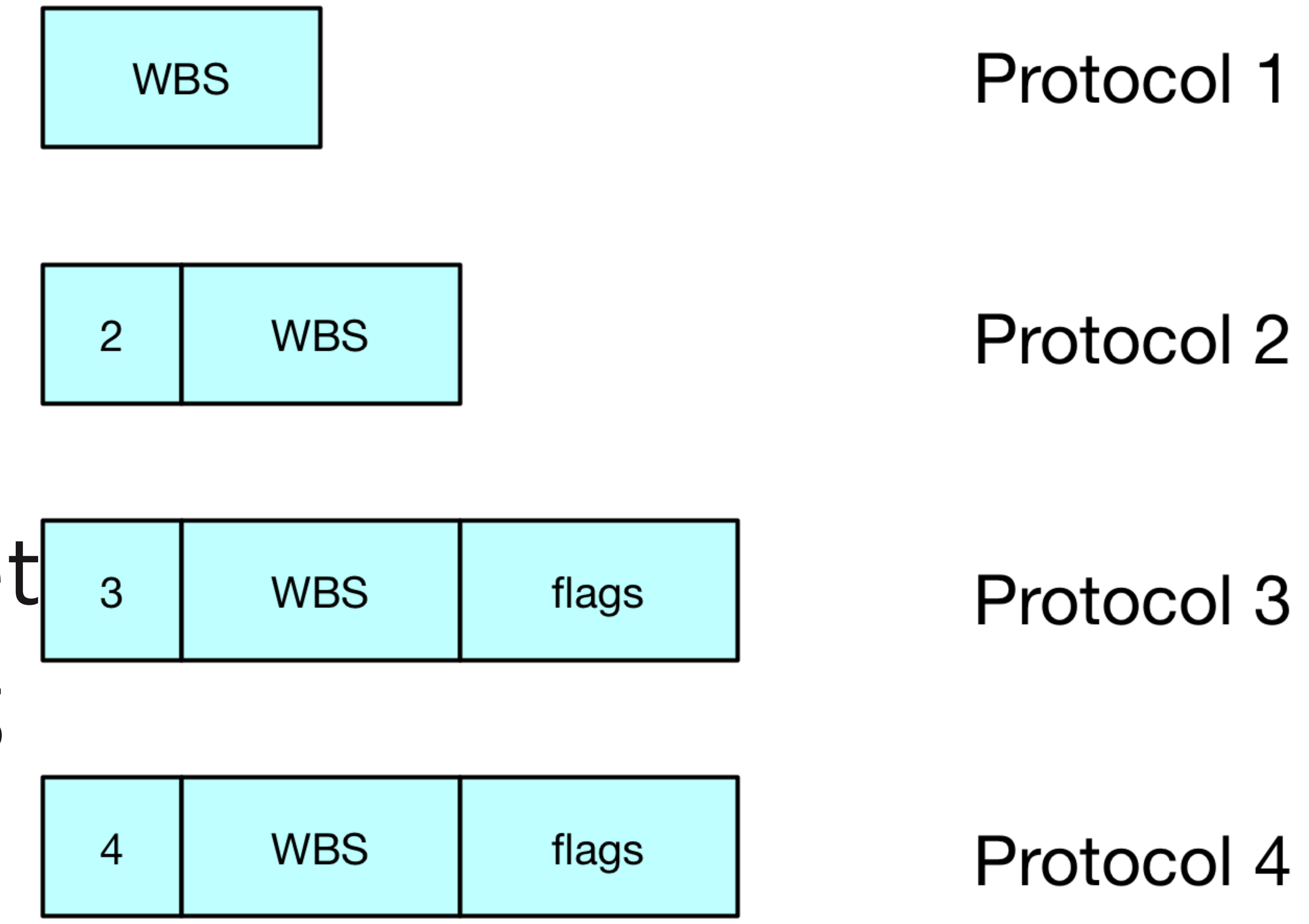

AFU Firmware Upgrade Protocol

- a HID Get Feature Report 0xB8 is sent to find out which of the 4 AFU protocols is spoken and how big the writebuffer (WBS) is

```
fprintf(f, "getting report: 0xb8\n");
rlen = sizeof(buffer);
memset(rbuffer, 0, sizeof(rbuffer));
kr = IOHIDDeviceGetReport(hiddevice, kIOHIDReportTypeFeature, 0xb8, &rbuffer, &rlen);

if (kr != 0) { _exit(1); }
```

- protocol 1 and 2 always send offsets
- protocol 3 and 4 send offsets if flags bit 1 is not set
- protocol 3 and 4 use extended setup it flags bit 0 is set
- protocol 1 and 4 use different HID reports than 2 and 3



AFU Firmware Upgrade Protocol

- a HID Set Feature Report 0xB0 with the magic code 0xc3 0xbc is sent if extended setup is required by protocol

```
if (CONFIG_need_extended_setup) {

    printf("setting report: 0xb0\n");
    rlen = sizeof(buffer);
    memset(rbuffer, 0, sizeof(rbuffer));

    rlen = 3;
    rbuffer[0] = 0xb0;
    rbuffer[1] = 0xc3;
    rbuffer[2] = 0xbc;

    kr = IOHIDDeviceSetReport(hiddevice, kIOHIDReportTypeFeature, 0xb0, &rbuffer, rlen);

}
```


AFU Firmware Upgrade Protocol

- a HID Set Feature Report 0xB0 with the magic code 0x62 0x72 is sent to start the firmware upload

```
fprintf(f, "setting report: 0xb0\n");
rlen = sizeof(buffer);
memset(rbuffer, 0, sizeof(rbuffer));

rlen = 3;
rbuffer[0] = 0xb0;
rbuffer[1] = 0x62;
rbuffer[2] = 0x72;

kr = IOHIDDeviceSetReport(hiddevice, kIOHIDReportTypeFeature, 0xb0, &rbuffer, rlen);
```


AFU Firmware Upgrade Protocol

- the firmware is sent piece by piece by sending HID Set Feature Report 0xB1
- if the protocol allows (requires) offsets to be sent the first 3 bytes of the buffer are the big endian offset

```
fprintf(f, "setting report: 0xb1\n");
rlen = sizeof(buffer);
memset(rbuffer, 0, sizeof(rbuffer));

int datalen = AFU_writebuffersize-1-3;
int datapos = 1;

rbuffer[0] = 0xb1;
if (CONFIG_send_offsets) {
    rbuffer[3] = offset & 0xFF;
    rbuffer[2] = (offset >> 8) & 0xFF;
    rbuffer[1] = (offset >> 16) & 0xFF;
    datapos += 3;
}
memcpy(&rbuffer[datapos], DATA+offset, datalen);
rlen = datapos + datalen;
offset += datalen;

kr = IOHIDDeviceSetReport(hiddevice, kIOHIDReportTypeFeature, 0xb1, &rbuffer, rlen);
```


AFU Firmware Upgrade Protocol

- the firmware is committed by sending HID Set Feature Report 0xB2 with code 0x00

```
fprintf(f, "setting report: 0xb2\n");
rlen = sizeof(buffer);
memset(rbuffer, 0, sizeof(rbuffer));

rlen = 2;
rbuffer[0] = 0xb2;
rbuffer[1] = 0;

kr = IOHIDDeviceSetReport(hiddevice, kIOHIDReportTypeFeature, 0xb2, &rbuffer, rlen);
```

AFU Firmware Upgrade Protocol

- a HID Get Feature Report 0xB2 is sent to retrieve information about success of AFU upload
- error code 0xA1 says success other error codes might reveal a broken signature

```
fprintf(f, "getting report: 0xb2\n");
rlen = sizeof(buffer);
memset(rbuffer, 0, sizeof(rbuffer));
kr = IOHIDDeviceGetReport(hiddevice, kIOHIDReportTypeFeature, 0xb2, &rbuffer, &rlen);
```


AFU Firmware Upgrade Protocol

- if extended setup is required by the protocol an advanced commit is required by sending HID Set Feature Report 0xB2 with code 0xc3 0xbc

```
fprintf(f, "setting report: 0xb2\n");
rlen = sizeof(buffer);
memset(rbuffer, 0, sizeof(rbuffer));

rlen = 3;
rbuffer[0] = 0xb2;
rbuffer[1] = 0xc3;
rbuffer[2] = 0xbc;

kr = IOHIDDeviceSetReport(hiddevice, kIOHIDReportTypeFeature, 0xb2, &rbuffer, rlen);
```

AFU Firmware Upgrade Protocol

- the firmware upgrade is finished by resetting the device by sending a HID Set Feature Report 0xB3

```
fprintf(f, "setting report: 0xb3\n");
rlen = sizeof(buffer);
memset(rbuffer, 0, sizeof(rbuffer));

rlen = 1;
rbuffer[0] = 0xb3;

kr = IOHIDDeviceSetReport(hiddevice, kIOHIDReportTypeFeature, 0xb3, &rbuffer, rlen);
```

DEMO

- Downgrade of Apple Pencil from 2.48 to 2.40
  (Apple forgot a downgrade protection in the Pencil)

- could be useful to downgrade to a firmware with a vulnerability
- then exploit the vulnerability and takeover the pencil

Apple Smart Keyboard SPI Storage/Retrieval

Apple Smart Keyboard SPI Storage

- when firmware is uploaded via HID Set Feature Report 0xB1 it is written via SPI
- unknown where it is actually written - but non volatile storage
- write access is forced to be within 0x20000 and 0x3b000
- depending on firmware type this data copied via SPI after commit and verification
  - firmware type 0xd0 is copied to 0x1e000
  - firmware type 0xc0 is copied to 0x3b000

Apple Smart Keyboard SPI Retrieval

- Apple Smart Keyboard supports HID Feature Report 0xB6 for SPI retrieval
  - HID Get Feature Report 0xB6 reads from current offset via SPI
  - HID Set Feature Report 0xB6 allows to set current offset
- storage seems to be non volatile
- whatever was previously written can be read back

- ATTENTION:
  on Apple Smart Keyboard 10.5“ this feature is broken and **returns uninitialised data**

Apple Smart Keyboard SPI Infoblock

- at offset 0x1e000 there is an info block that contains information about the keyboard
- serial numbers of attached keyboard, product id, vendor id, hardware revision, …

```
0001e000  f0 9f 92 b0 00 01 ac 05  6a 02 00 00 37 f5 55 59  |........j...7.UY|
0001e010  00 00 11 0c 46 54 50 54  57 31 43 43 48 43 35 31  |....FTPTW1CCHC51|
0001e020  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
0001e030  00 00 00 00 46 50 57 37  32 30 37 30 44 37 48 48  |....FPW72070D7HH|
0001e040  50 4c 48 41 59 00 00 00  00 00 00 00 00 00 00 00  |PLHAY...........|
0001e050  00 00 00 00 ff ff ff ff  ff ff ff ff ff ff ff ff  |................|
0001e060  ff ff ff ff ff ff ff ff  ff ff ff ff ff ff ff ff  |................|
```


Apple Smart Keyboard Firmware Dumping (I)

- we can use HID Feature Request 0xB6 to read whatever firmware was last uploaded
- let’s assume that Apple uses the same feature to upload the final firmware of a device
- what if we buy a new keyboard and try to read what was last uploaded
- between 0x20000 and 0x3b000 there should be the firmware

Apple Smart Keyboard Firmware Dumping (II)

- unfortunately Apple does first upload the firmware and then the infoblock
- this means the first 4kb of the firmware is destroyed when extracted this way

```
00020000  c7 a2 00 01 d0 00 00 00  54 00 00 00 c3 c7 52 25  |........T.....R%|
00020010  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
00020070  00 00 00 00 00 00 00 00  00 00 00 00 a4 26 58 a3  |.............&X.|
00020080  f0 9f 92 b0 00 01 ac 05  6a 02 00 00 37 f5 55 59  |........j...7.UY|
00020090  00 00 11 0c 46 54 50 54  57 31 43 43 48 43 35 31  |....FTPTW1CCHC51|
000200a0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
000200b0  00 00 00 00 46 50 57 37  32 30 37 30 44 37 48 48  |....FPW72070D7HH|
000200c0  50 4c 48 41 59 00 00 00  00 00 00 00 00 00 00 00  |PLHAY...........|
000200d0  00 00 00 00 ff ff ff ff  ff ff ff ff ff ff ff ff  |................|
000200e0  ff ff ff ff ff ff ff ff  ff ff ff ff ff ff ff ff  |................|
*
00021000  05 05 b1 f8 00 c0 1c f0  ff 0f 35 d0 14 68 4f f0  |..........5..hO.|
```

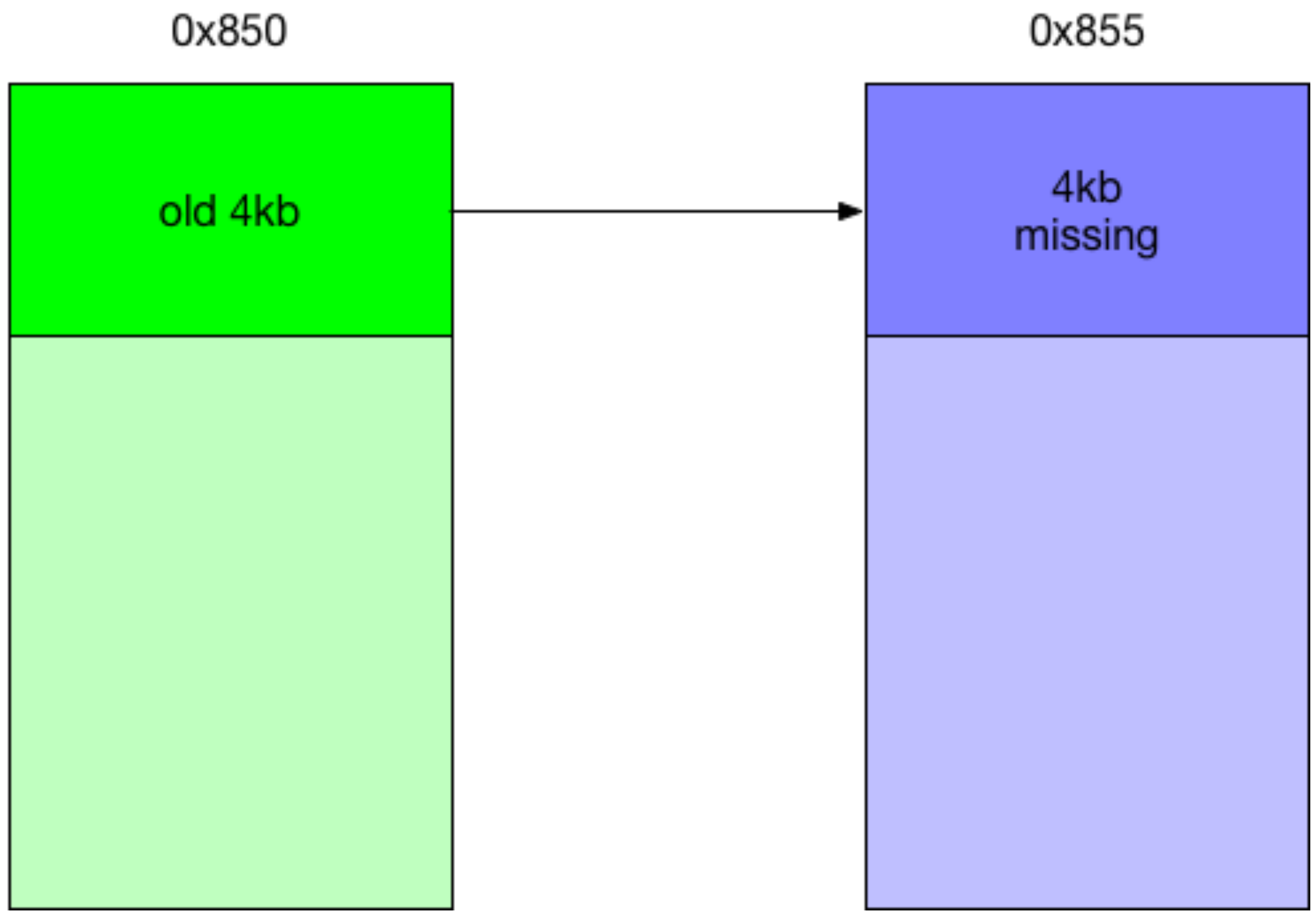

Apple Smart Keyboard Firmware Reconstruction

- for testing we dumped the firmware of a fresh
  - Apple Smart Keyboard 12.9“ - version 0x855
  - Apple Smart Keyboard 9.7“ - version 0x858
- we only have Apple Smart Keyboard 12.9“ - version 0x850 available for reconstruction

 |

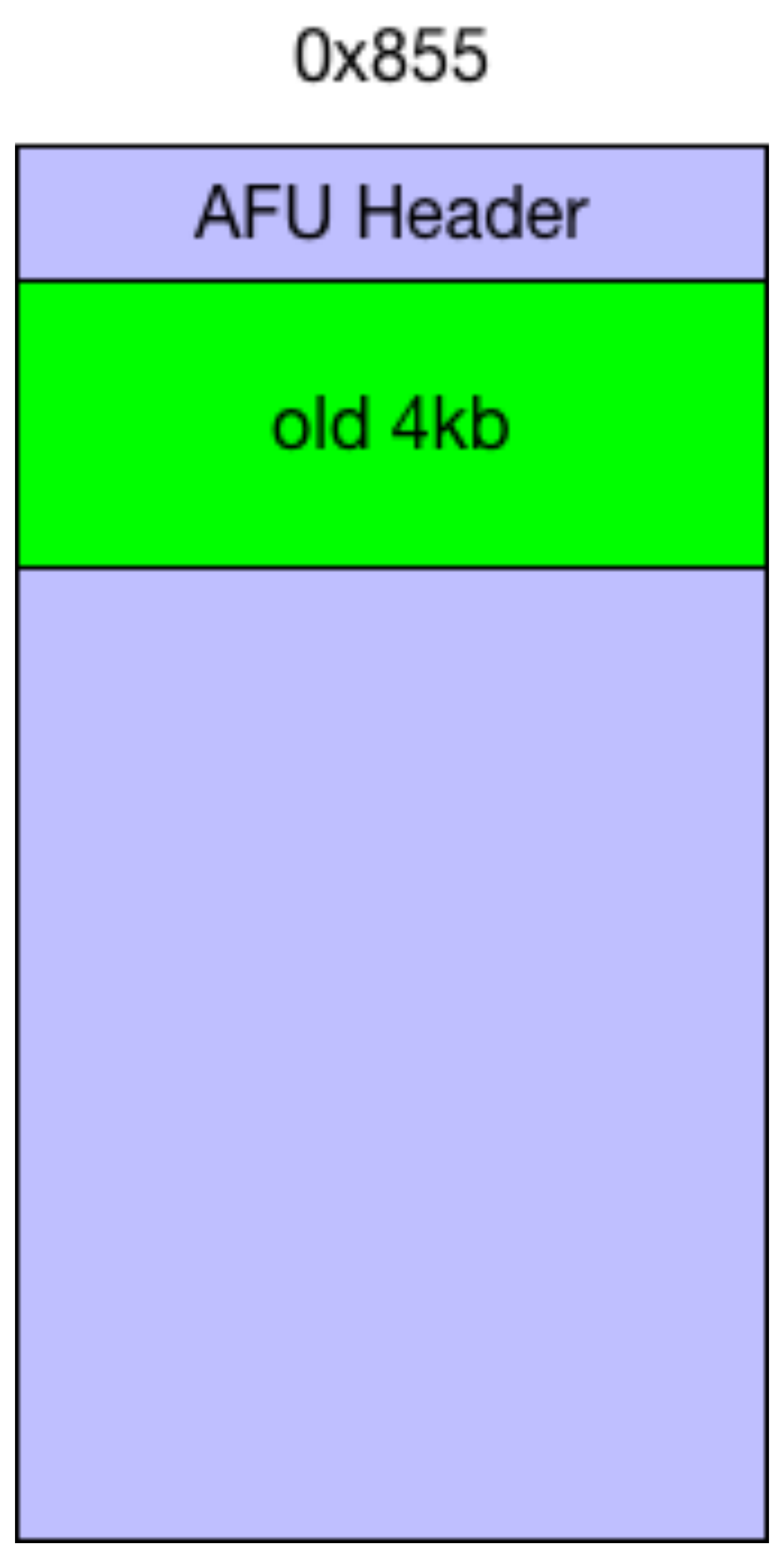
Apple Smart Keyboard Firmware Reconstruction (II)

- we can copy the first 4kb from 0x850 to 0x855 because we can see that around 4kb firmwares are identical



Apple Smart Keyboard Firmware Reconstruction (III)

- we can always reconstruct a valid AFU header because purpose of all fields is determined

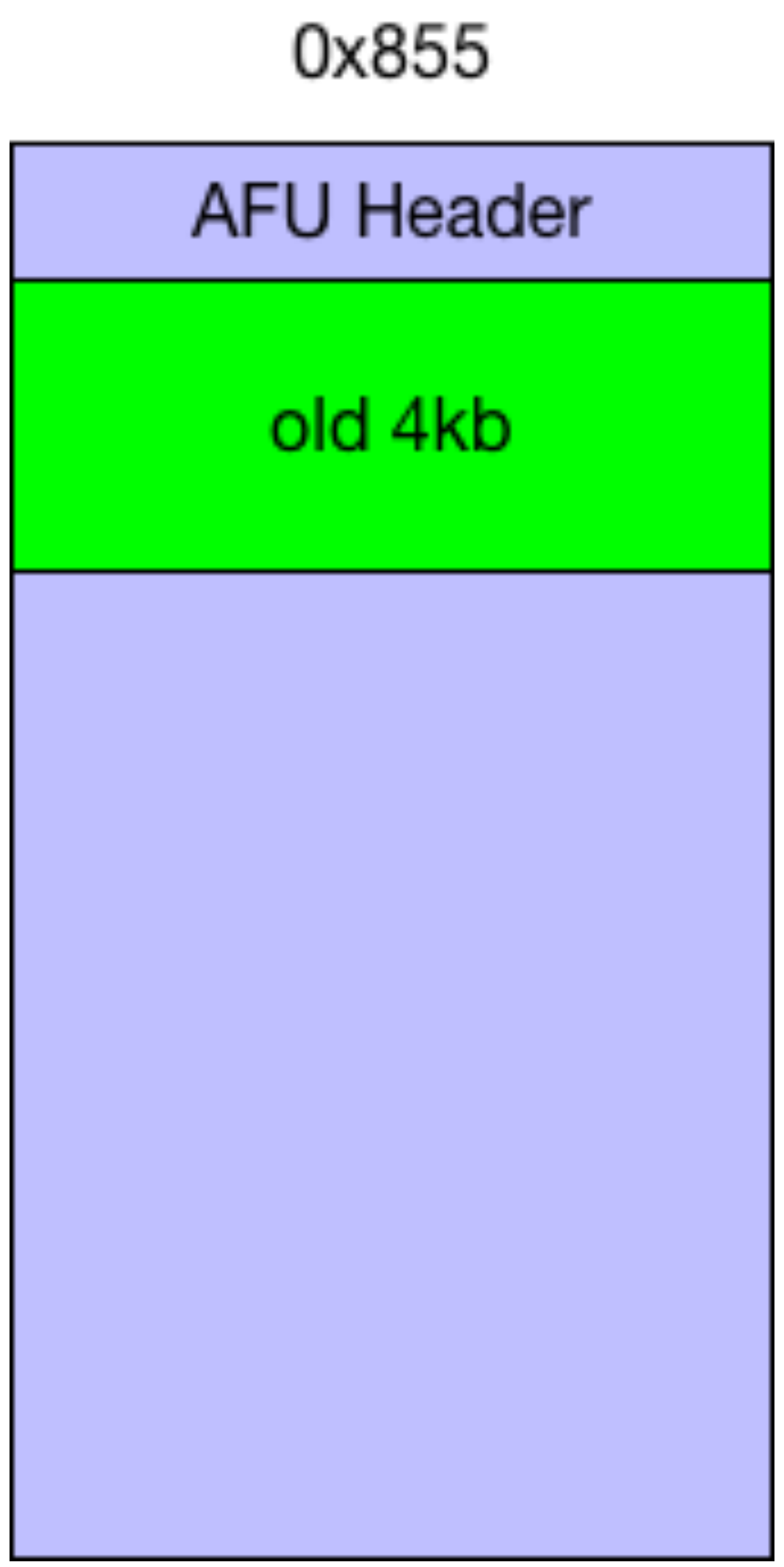


Apple Smart Keyboard Firmware Reconstruction (IV)

- at this point we assumed that there were no actual code changes !!!
- this means most of the 4kb should be the same

- except for:
  - version number inside green
  - 2 label relative pointers from green to blue
  - pointer to blue inside green
  - relative jump from green to blue with changed target address

  - pointer to SRAM from green (and SRAM addressed changed)
    (luckily this did not occur)



Apple Smart Keyboard Firmware Reconstruction (V)

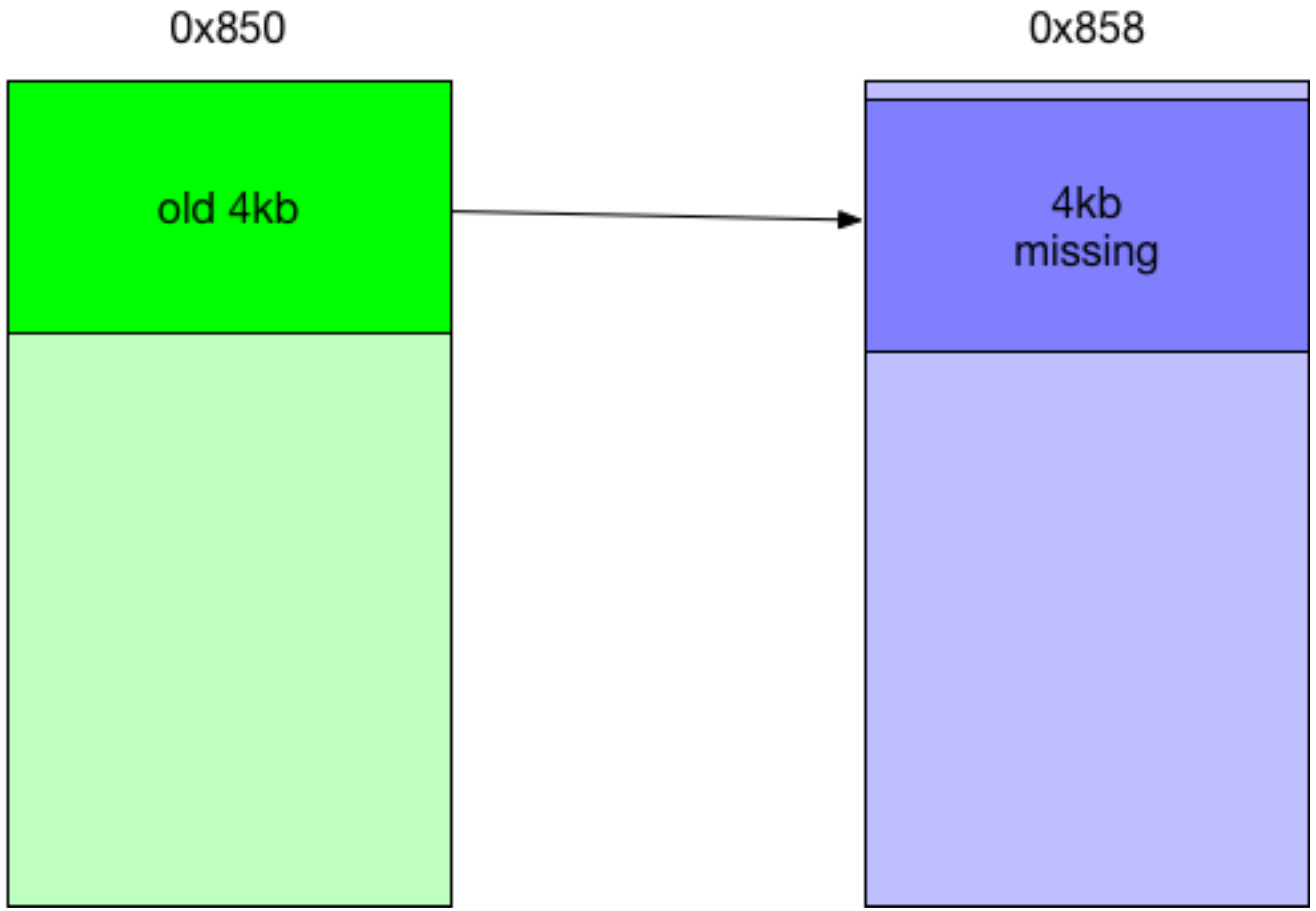
- luckily there were not that many changes needed
- it was possible to fully reconstruct the firmware
- even the RSA2048 signature matches

```
$ python ../AFU_check.py B249-STFW-0x0855-prod-signed.bin

[+] Checking AFU file B249-STFW-0x0855-prod-signed.bin
[+] AFU magic matches
[+] AFU for Smart Keyboard 12.9"
[!] CRC algorithm: standard
[+] AFU header CRC matches
[+] AFU data CRC matches
[+] AFU sha256 digest matches
[+] AFU RSA2048 signature matches
```

Apple Smart Keyboard Firmware Reconstruction (VI)

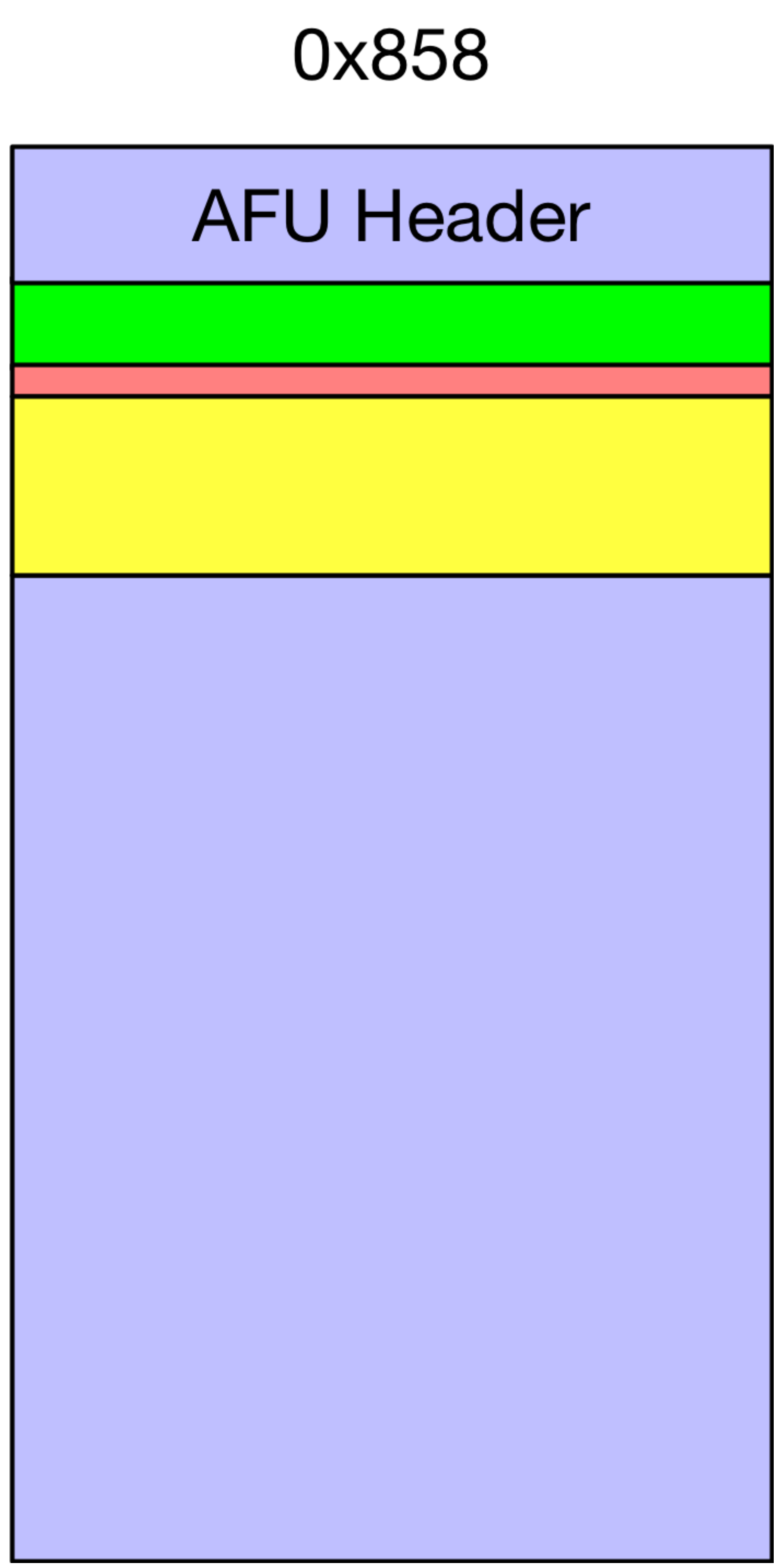
- the 0x858 firmware from the Apple Smart Keyboard 9.7“ was a different beast
- we realized that around 4 kb the code was very much the same but it was shifted by 16 bytes



 |

Apple Smart Keyboard Firmware Reconstruction (VII)

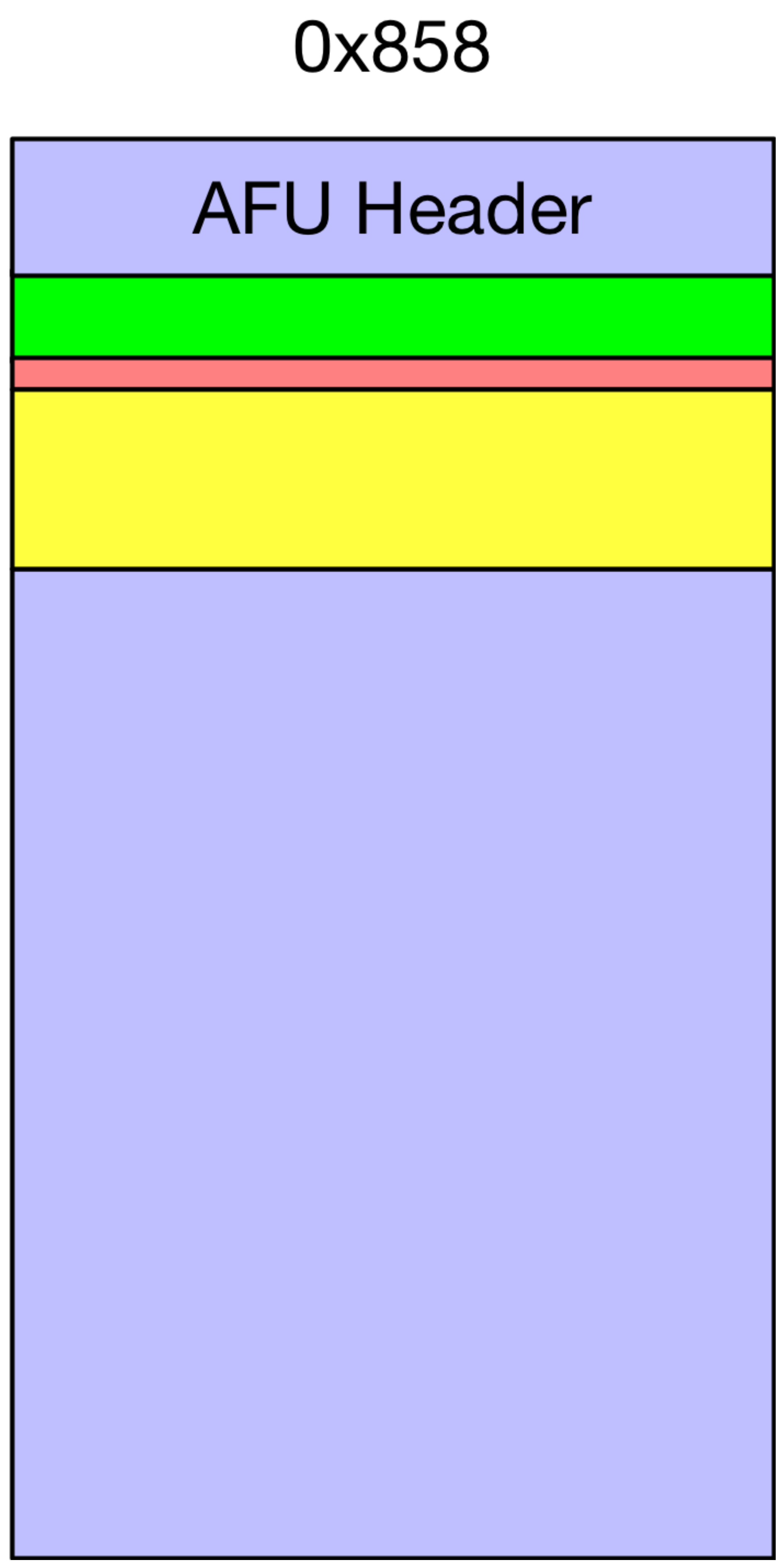
- it was easy to pinpoint where the shifted 16 bytes occurred
- it seems one 16 byte function was introduced within the first 4kb
- we have no idea what these 16 bytes are
- but if we ignore these 16 bytes we can reconstruct everything else
- of course we cannot make the hash/signature match
- but it is good enough to analyse



Apple Smart Keyboard Firmware Reconstruction (VIII)

- we now have to deal with:
  - version number inside green
  - 2 label relative pointers from green to blue
  - pointer to SRAM from green (and SRAM addressed changed)
  - pointer to blue inside green or yellow
  - relative jump from green to blue with changed target address
  - relative jump from blue to green because of 16 byte shift
  - relative jump from yellow to green because of 16 byte shift

➡ we have something reconstructed that works great in IDA



Apple Pencil Crashlog

Apple Pencil Crashlog

- Apple Pencil contains debugging tool the Crashlog
- panics will be written there
- might be useful during exploitation of vulnerabilities
- access to crashlog via HID Feature Report 0xE1

HID Set Feature Report 0xE1

- crashlog can be controlled with HID Set Feature Report 0xE1
- first data byte selects sub function
  - `0x05` – set internal crashlog read offset (followed by 2 bytes little endian offset)
  - `0xee` – erase the crashlog content
  - `0xcc` – create a test crashlog entry

HID Get Feature Report 0xE1

- crashlog can be read with HID Get Feature Report 0xE1
- reads from current crashlog read offset
- offset is automatically increased

 |

Apple Pencil Crashlog

```
Crash log for exception 0x10
Crash log test!@src/target/b222/target_hid.c:996
taskID 6
numTasks 11
freeHeap 1248
 0 Idle task P127 St:0 Stk:0/0(0%)
 1 <task(1)> P60 St:80 Stk:160/376(43%)
 3 <task(3)> P126 St:1 Stk:88/256(34%)
 4 PowerMgr P24 St:40 Stk:648/1096(59%)
 5 radio_interface P64 St:40 Stk:696/1200(58%)
*6 idbus P50 St:0 Stk:1272/1696(75%)
 8 Force P31 St:40 Stk:688/1200(57%)
 9 Inertia P63 St:40 Stk:1072/1800(60%)
 10 battery P123 St:40 Stk:632/984(64%)
 11 iap_idbus P100 St:40 Stk:680/880(77%)
 12 BTSync P20 St:40 Stk:448/1024(44%)
bt, size 84 taskId 6**, err 0x0, depth 14, idx 0
=>0x20006a50
  0x20000e14
  0x8031302
  0x8021bdd
  0x0
  0xf0
  0x0
  0x9f
  0x0
  0x20001710
  0x200016c0
  0x200016c0
  0x88888806
  0x0
FW_v248...2017/07/26...22:19:06 on 323151.A0 – protocol: 1.0.0
```


Results and Conclusion

Results - Firmware Upgrade Process

- PRO
  - firmware upgrades use RSA2048
  - it seems the validation cannot be tricked
  - Apple Smart Keyboard is protected against firmware downgrades
- CON
  - Apple forgot firmware downgrade protection in Apple Pencil
  - old firmware stays in Apple Smart Keyboard storage and can be read back
- QUESTION
  - does the bootloader check sig too? Or does ONE TIME CODE EXEC defeat the system?

Results - Security Architecture

- CON
  - we did not see any exploit mitigations (e.g. ASLR, Canaries, NX)
  - Apple Smart Keyboard does not seem to even have MPU support
  - Apple Pencil does not seem to use MPU
  - sandboxed iOS Applications can talk to accessory and even downgrade firmware
  - why is **fud** not sandboxed?

Conclusion

- we found a number of smaller problems
- but no dramatic problem like code execution(, yet)

- but we have only looked at a part
- especially the Apple Pencil has parsers we haven’t fully understood and checked, yet (bluetooth, IAP, …)
- we also want to do some hardware attacks
- and try to get the firmware running in a QEMU STM32 port

Questions?

www.antid0te.com