Antid0te SG

Introduction to the Security of Apple Pencil and Apple Smart Keyboard

revision 0.9

Stefan Esser <stefan@antid0te-sg.com>

Table of Contents

1. Apple Pencil and Apple Smart Keyboard

Research into the security of the Apple Pencil and the Apple Smart Keyboard was motivated by the question how secure these accessories are that we connect to our iOS devices on a daily basis. The security model of iOS with its code signed boot chain, encrypted data storage and process sandboxing functionality is very well understood and has been documented over the years by Apple and in lots of 3rd party research projects. However to the best of our knowledge no such public evaluation of the security of Apple Pencil and Apple Smart Keyboard exists.

With this paper that is planned to be the first of a number of papers about this topic we want to start the public research into the security of these accessories. We therefore concentrate on the following questions:

1. what CPU is running in these devices?
2. how to get the firmware of these devices?
3. how is this firmware packaged and/or signed?
4. what is inside this firmware?
5. how do firmware updates work?
6. how is updated firmware verified?

1.1. IFIXIT Teardowns

A good start for research into devices like the Apple Smart Keyboard and the Apple Pencil is to check what is actually inside these products in terms of microcontrollers and other chips. A good resource for that is the IFIXIT website and their product teardowns in which they open up the products, usually with the goal to judge repairability and document on the way what they find inside.

The IFIXIT teardown of the Apple Smart Keyboard 12.9" can be found at:
https://www.ifixit.com/Teardown/Smart+Keyboard+Teardown/53052

The teardown comes to the conclusion that the microcontroller used within is the:
ST Microelectronics STM32F103VB 72 MHz 32-bit RISC ARM Cortex-M3

NOTE: There has been no teardown of the Apple Smart Keyboard 9.7" and 10.5"

The teardown of the Apple Pencil is available at:
https://www.ifixit.com/Teardown/Apple+Pencil+Teardown/52955

This teardown concludes that the Apple Pencil uses an:
ST Microelectronics STML151UCY6 Ultra-low-power 32-bit RISC ARM Cortex-M3

1.2. STM Microcontrollers

The fact that Apple uses standard STM microcontrollers for the Apple Pencil and the Apple Smart Keyboard is good for firmware reverse engineering purposes, because STM has very good documentation of their products with data sheets and programming guides available for free from their website.

Apple Pencil (STM151UCY6)

Product URL: https://www.st.com/en/microcontrollers/stm32l151c6.html

Datasheet: https://www.st.com/resource/en/datasheet/stm32l151cc.pdf

ARM Cortex-M3

SRAM: 32 Kbytes

Flash memory: 256 Kbytes

Data EEPROM: 8 Kbytes

Option bytes: for write-protect, read-out protection, 4 Kbytes granularity

Memory Protection Unit: YES

Apple Smart Keyboard (STM32F103VB)

Product URL: https://www.st.com/en/microcontrollers/stm32f103vb.html

Datasheet: https://www.st.com/resource/en/datasheet/stm32f103vb.pdf

ARM Cortex-M3

SRAM: 20 Kbytes

Flash memory: 128 Kbytes

Memory Protection Unit: **NO**

2. Accessory Firmware Update

Having answered the question what microcontrollers are used in the accessories of interest we need to figure out where to get a copy of the firmware running on these accessories. A good source for such a copy is usual to download one of the firmware updates from Apple. However these updates are not provided as normal download binaries on the Apple website. Instead detection and retrieval of these updates happens in the background. *The iPhone Wiki*[1] is usually a good starting point for these inquiries. In our case the information we need is already partially documented on the *OTA_Updates*[2] page. On that page we can find links to the Accessory Software Update XMLs and it also leads to an archive of previously provides firmware updates for the Apple Pencil and the Apple Smart Keyboard (12.9")

2.1. Accessory Firmware Update Manifest Retrieval URLs

Automatic firmware retrieval starts with the system first fetching XML files from Apple's servers that describe what firmware updates are available. These XML files are in a fixed location that is derived from the type of accessory that needs to update its firmware. For our purposed the official XML file URLs are as follows:

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

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

---

[1] The iPhone Wiki: https://www.theiphonewiki.com/

[2] OTA Updates: https://www.theiphonewiki.com/OTA_Updates

At the time of writing the XML Retrieval URL for the Apple Smart Keyboard 9.7" point to an Accessory Update XML file that lists no firmware update and the XML Retrieval URL for the Apple Smart Keyboard 10.5" even results in an access denied error XML. In both cases it is important to know that Apple has not provided any public firmware updates for any of these devices, yet.

2.2. Accessory Firmware Update Manifest

The XML retrieved from Apple describes what firmware updates are available for the given accessory. This includes the version number, the download URL, download size, SHA-1 hash of the firmware bundle and archive format that is usually ZIP. Furthermore the XML seems to be signed with the AssetManifestSigning key, that comes within the certificate field of the XML. A human readable form of the certificate used is in the appendix of this document. The interesting thing about this certificate is that it expired on 14th July 2018 and Apple has not updated it, yet. It is unknown at this point if this would stop firmwares from ever being updated on newly restored devices.

```
<?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/>
        <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>
```

2.3. Accessory Firmware Update ZIP Files

The firmware update bundle downloaded from Apple is usually packaged inside a ZIP file. The content of the ZIP file slightly varies because different accessories have different firmware types packed into the same bundle.

Apple Smart Keyboard

A firmware update bundle for the Apple Smart Keyboard usually has this directory layout:

```
./META-INF/com.apple.ZipMetadata.plist
./AssetData/B249-STFW-0x0850-prod-signed.bin
./Info.plist
```

It contains the usual bundle plist files and a `.bin` file that is actually an `.afu` file. The file format of these AFU files is described in the next part of this document.

Apple Pencil

A firmware update bundle for the Apple Pencil usually contains a few more files and has a directory layout similar to the following:

```
./META-INF/com.apple.ZipMetadata.plist
./AssetData/B222_FW_v0246_EVT.prod-sig.afu
./AssetData/B222_FW_v0246_Final.prod-sig.afu
./Info.plist
```

In this example the firmware update bundle contains 2 AFU files. A comparison of the two files shows that they are for two different hardware revisions of the Apple Pencil. The EVT file is for a hardware revision 0xE and the Final file is for a hardware revision of 0xF. Aside from that only CRCs, digest and signature are different. For more information about the AFU file format see the next chapter.

3. AFU File Format

Firmware updates for Apple Pencil, Apple Smart Keyboard and other Accessories like the Siri Remote are provided in what is called an AFU file. However the file extension used by these files is not necessarily `.afu` and could be something else like `.bin` or `.sig`. To the best of our knowledge it is not documented what the abbreviation AFU really stands for, but **A**ccessory **F**irmware **U**pdate fits perfectly.

3.1. AFU File Header

AFU files start with an 128 byte header that contains general information about the firmware like what type of firmware it is, what version it is and what product in what hardware revision it supports. The header itself also contains the CRC of header and the length of the firmware data and a CRC of that data. Within the header at offset `0x20` there can be an optional signature header and at offset `0x40` there can be an optional personalization header. The exact layout of the header is shown in the following table.

Table: AFU File Header Layout

Offset	Length	Content
0x00	2 bytes	AFU File Magic (0xC7 0xA2)
0x02	2 bytes	AFU File Version (0x100) - guess
0x04	2 bytes	Firmware type
0x06	2 bytes	Firmware version
0x08	4 bytes	File offset of Firmware Data
0x0C	4 bytes	CRC of Firmware Data
0x10	2 bytes	Product ID
0x12	2 bytes	Hardware Revision ID
...		
0x20	24 bytes	AFU Signature Header
...		
0x40	24 bytes	AFU Personalization Header
...		
0x7C	4 bytes	CRC of AFU Header bytes 0x00 .. 0x7C

Firmware Type

The following table of firmware types has been extracted from StandaloneHIDFudPlugins which might be an incomplete list of firmwares.

Table: List of Firmware Types

Code	Constant
0x01	kAUTypeSTFW
0x20	kAUTypeMTFW
0x30	kAUTypeRadioFW
0x31	kAUTypeRadioDiags
0x40	kAUTypeAudioFW
0x41	kAUTypeAudioCalFW
0x50	kAUTypeChargerFW
0xa0	kAUTypeVibeWaveform
0xb0	kAUTypeBootLoader
0xc0	kAUTypeKeyCal
0xc1	kAUTypeMTCal
0xc2	kAUTypeForceCal
0xc3	kAUTypeActCal
0xc4	kAUTypeAccelCal
0xc5	kAUTypeAudioCal
0xdf	kAUTypeTest

CRC

The CRC algorithm used for calculating the CRC values inside the header depends on the product used. Most products seem to use the standard CRC32 format, but the Apple Pencil uses the built-in STM32 CRC functionality that produces a non standard CRC.

Product ID and Hardware Revision ID

AFU files are used for upgrading firmwares of different types of products it is therefore necessary to define inside the header what specific product and what hardware revision of the product a firmware is for. The following table lists a number of product identifiers that are known to us.

Table: Product IDs

Product ID	Product
0x222	Apple Pencil (firmware for 0x312 allows this product id, too)
0x266	Apple Siri Remote 1
0x268	Apple Smart Keyboard (12.9”)
0x26a	Apple Smart Keyboard (9.7”)
0x26b	Apple Smart Keyboard (10.5”)
0x26d	Apple Siri Remote 2
0x312	Apple Pencil

3.2. AFU Signature Header

Firmware stored in AFU files is usually required to be signed. In order to be signed an AFU Signature Header must be added to offset `0x20` of the AFU File Header. The format of this header is described in the following table.

Table: AFU Signature Header Layout

Offset	Length	Content
0x00	4 bytes	AFU Signature Magic (0x24 0x47 0xe3 0x61)
0x04	2 bytes	AFU Signature Header Version (0x100)
0x06	2 bytes	Unknown 0x120 - maybe size of digest plus size of signature
0x08	2 bytes	Digest Type (1: sha256)
0x0A	2 bytes	Digest Length
0x0c	4 bytes	Digest File Offset
0x10	2 bytes	Signature Type
0x12	2 bytes	Signature Length
0x14	4 bytes	Signature File Offset

Digest

At the moment firmware for the Apple Smart Keyboard and the Apple Pencil only support the Digest Type `0x01` which is SHA256. The length is therefore 32 bytes.

Signature

AFU Files currently use an RSA 2048 bit signature, which is a PKCS padded SHA256 digest. However two Signature Types exists at the moment that use two different RSA public keys for verification of the firmware. From the strings of the Apple Pencil firmware it is believed that type 2 is a key used for personalized firmware and type 3 is a key used for a normal production firmware.

3.3. AFU Personalization Header

From reversing the firmware of the Apple Smart Keyboard it was discovered that firmware can be personalized for a specific device by embedding an AFU Personalization Header at offset `0x40` inside the AFU File Header. This header includes a unique id of the device and locks the signature onto this one specific device. At the time of writing only the Apple Smart Keyboard seems to support this, the Apple Pencil doesn't.

Table: AFU Personalization Header Layout

Offset	Length	Content
0x00	4 bytes	AFU Personalization Magic (0x82 0x06 0x4a 0x33)
0x04	2 bytes	AFU Personalization Header Version (0x100)
0x06	X bytes	UNIQUE ID
0x1c	4 bytes	Flags (1: relaxed verification of firmware parameters)

UNIQUE ID

At the time of writing only the Apple Smart Keyboard firmware supports the verification of the device UNIQUE ID. The UNIQUE ID used is the 96 bit value from the UNIQUE ID Registers that are located at 0x1FFFF7E8 in the STM32F103VB.

4. Firmware Content

When we take a look at the AFU files contained in the firmware bundles of Apple Pencil and Apple Smart Keyboard we can see a huge difference in size. While the Apple Pencil has a firmware update that is nearly 183 Kbytes in size the Apple Smart Keyboard has only a size of around 45.5 Kbytes. This is mostly because the Apple Pencil is more complex internally than the keyboard and because the firmware of the Apple Pencil has way more strings inside, which can help a lot during reversing the firmware.

We know both microcontrollers that are used within Apple Pencil and Apple Smart Keyboard are ARM Cortex-M3 based. This means among other things that they use a Thumb2 instruction set. Most disassemblers with ARM 32 bit support will therefore be able to correctly disassemble the firmware files. However none of the standard disassemblers will understand the AFU file format out of the box and therefore the only choice is to load the AFU files as binary files.

When loading AFU files as binary files we have to answer a number of questions:

1. is the header part of the address space?
2. is the content of the AFU file a flat address space?
3. what is the load base address of the binary file?
4. what is the entry point?

At this point we cannot really answer any of the two first questions because that would require further information that we do not have at this point. However because the AFU header is relatively simple and only contains one big data blob we start by assuming that AFU file contain a flat address space.

4.1. Entry Point and Base Address

For starters we just take a look at the data after we have stripped away the header. At this point we assume that what we are dealing with is the plain firmware for an ARM Cortex-M3 microcontroller. These firmwares usually start with the InitialMSP, which is the initial stack pointer after an reset, followed by the exception table. For an ARM Cortex-M3 this exception table is a list of pointers to the handlers. We also know that the handlers must be Thumb2 code.

When we take the Apple Smart Keyboard firmware as example we can see the initialMSP and the exception handler table right at the beginning of the data. The initialMSP points

into memory that is SRAM according to the memory map of the STM32F103VB[3]. The handlers point into the alias area of the Flash memory.

```
ROM:00000000                DCD 0x20002650  <- initialMSP
ROM:00000004                DCD 0x80027E5   <- Reset
ROM:00000008                DCD 0x8003759   <- NMI
ROM:0000000C                DCD 0x8003729      ...
ROM:00000010                DCD 0x8003755
ROM:00000014                DCD 0x8002F4F
ROM:00000018                DCD 0x80044C3
ROM:0000001C                DCD 0
ROM:00000020                DCD 0
ROM:00000024                DCD 0
ROM:00000028                DCD 0
ROM:0000002C                DCD 0x8003E4D
ROM:00000030                DCD 0x8003225
ROM:00000034                DCD 0
ROM:00000038                DCD 0x8003895
ROM:0000003C                DCD 0x8003E51
ROM:00000040                DCD 0x80027FF
ROM:00000044                DCD 0x80027FF
ROM:00000048                DCD 0x80027FF
```

From this information we can learn that the actual base address of the firmware should be somewhere between 0x08000000 and 0x080027E5, because the base address must be below the lowest handler address. So if we assume it is 0x08000000 we will end up with the following code as RESET vector:

```
ROM:080027D8                LDR             R1, [R0,#8]
ROM:080027DA                LDR             R0, =0x200006B8
ROM:080027DC                ADD.W           R0, R0, #5
ROM:080027E0                BL              sub_8000358
ROM:080027E4                LDR             R0, =0x200006B8
ROM:080027E4                                        ; DATA XREF: ROM:08000004↑o
ROM:080027E6                STRB            R5, [R0,#1]
ROM:080027E8                LDR             R0, =0x20000090
ROM:080027EA                LDR             R0, [R0]
ROM:080027EC                CMP             R0, #0xFF
```

It should be obvious that the RESET vector would point into the middle of an actual function an not to the beginning of a function. One possible way to determine the base address would be to determine all function addresses and then check repeatedly if the exception handlers start pointing all to function starts when we repeatedly increase the base address.

[3] Datasheet page 34 (https://www.st.com/resource/en/datasheet/stm32f103vb.pdf)

However we have taken a different route to find the base address that turns out to be way easier. This alternative way is based on the assumption that somewhere inside the firmware code a back reference to the beginning of the firmware is embedded as pointer. We therefore wrote a script that outputs all pointers inside the Flash alias area that are below the lowest exception handler address. The script produced the following output for the Apple Smart Keyboard firmware:

```
0x8000000
0x8000100
0x8002000
0x8002004
0x8002580
0x8002600
0x80026ed
```

Most of these values can be eliminated and we could now manually check for 0x08002000 and 0x08002580 and 0x08002600. The result of this was that the actual base address of the data of the Apple Smart Keyboard firmware is 0x08002600. However at this point the fact that 0x08002580 was also referenced from within the firmware was a big indicator that in memory the AFU header is actually still present, because it is exactly 128 bytes away. At this point it was also not clear if the other revisions of the Apple Smart Keyboard 9.7" and 10.5" have the same base address, because no firmware binary compatible with these devices is provided as update from Apple.

The same technique can be used with the Apple Pencil firmware file and we end up with these following base addresses:

Device	AFU Header Address	AFU Data Address
Apple Pencil	0x08006000	0x08006080
Apple Smart Keyboard	0x08002580	0x08002600

With the base address known determining the entry point into the firmware is as simple as starting at the RESET vector. Of course all the other exception handlers can be used as additional entry points.

From the firmware base addresses we can also learn that the updatable firmware does not start at the beginning of the flash alias area. This likely means that there is some boot code located before the AFU firmware data that cannot be updated. At the time of writing this document it was not yet possible to read or dump this area and therefore we have no information about the content, yet.

For easier handling of AFU firmware updates an IDA Pro python plugin was developed that implements an AFU file loader. At the time of publication of this white paper this code will be available on GitHub at https://github.com/antid0tecom/ipad_accessory_research.

4.2. Finding the AFU File Format Parser

The firmware for accessories is usually quite big so there is a lot to reverse engineer until a whole picture can be understood. However in this paper we want to concentrate on the Accessory Firmware Update, the parts involved and especially the firmware verification on the accessory side. Therefore we need to find the AFU file format parser inside the firmware. Without any knowledge of the structure of the firmware there are usually two common methods to find a specific part of code:

1. find strings that hint to file format parsing
2. find code that uses constants used by the file format

The Apple Smart Keyboard has only very few strings inside its firmware. The Apple Pencil on the other hand has a few hundred strings inside, including file names. However none of the strings hints to an AFU file parser. We therefore tried to search for the constants that we know are part of the AFU file format. These constants are:

1. `0xA2C7` or `0xC7 0xA2`
2. `0x61E34724` or `0x24 0x47 0xE3 0x61`
3. `0x334A0682` or `0x82 0x06 0x4A 0x33`

However we are looking into ARM Thumb2 code here so these constants might either be stored as constants that are loaded via a PC relative LDR instruction or they are constructed with MOVW and MOVT instructions. Sometimes disassemblers like IDA even hide this from the user by faking a 32 bit MOV instructions. Here are examples of these possibilities.

```
fake 32 bit MOV instruction
__TEXT:08016F58                 LDR             R1, [R0]
__TEXT:08016F5A                 MOV             R2, #0x334A0682
__TEXT:08016F62                 CMP             R1, R2

separate MOVW/MOVT instructions
__TEXT:08016F0C                 MOV             R1, R0
__TEXT:08016F0E                 MOVW            R2, #0x4724
__TEXT:08016F12                 LDR             R0, [R1]
__TEXT:08016F14                 MOVT.W          R2, #0x61E3
__TEXT:08016F18                 CMP             R0, R2
```

```
constant load via PC relative LDR
__TEXT:08006C6C                 MOV             R1, R0
__TEXT:08006C6E                 LDR             R2, =0x61E34724
__TEXT:08006C70                 LDR             R0, [R1]
...
__TEXT:08006C80 dword_8006C80   DCD 0x61E34724
```

With these possibilities in mind it is relatively easy to pin point code areas inside the Apple Pencil or the Apple Smart Keyboard firmware that deal with the AFU signature. This will be discussed in detail in the chapter about the AFU Firmware Verification.

When we checked what code uses the AFU file format parsing code we realized that this happens in one big switch statement that seems to parse some incoming data. With more information from the code running on the iPad we realized that these switch statements have something todo with the parsing of HID Feature Reports.

5. Accessory HID Feature Reports

During our research it was discovered that the code that performs firmware updates on the iPad side makes use of the `StandaloneHIDFudPluging.bundle` that is located in `/System/Library/AccesoryUpdaterBundles/`. From reversing this bundle we learned that the firmware updates usually happen by sending special HID Feature Get/Set Reports commands to the HID Devices registered by the Apple Pencil and Apple Smart Keyboard. The report numbers involved also turned out to be the same numbers checked for in the big switch statement that was discussed in the previous section.

The IOKit registry shows that both Apple Pencil and Apple Smart Keyboard register a number of HID devices that the iPad can connect to and talk to:

```
+-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>
```

```
+-o Apple iPad Pro Smart Keyboard  <class IOAccessoryIDBusBulkData>
  +-o IOAccessoryIDBusBulkDataEndpoint@0  <class IOAccessoryIDBusBulkDataEndpoint>
  | +-o IOAccessoryIDBusHIDDevice  <class IOAccessoryIDBusHIDDevice>
  |   +-o IOHIDInterface  <class IOHIDInterface>
  |     +-o AppleHIDKeyboardDeviceManagement  <class AppleHIDKeyboardDeviceManagement>
  |       +-o IOHIDEventServiceUserClient  <class IOHIDEventServiceUserClient>
  +-o IOAccessoryIDBusBulkDataEndpoint@1  <class IOAccessoryIDBusBulkDataEndpoint>
  | +-o IOAccessoryIDBusHIDDevice  <class IOAccessoryIDBusHIDDevice>
  |   +-o IOHIDInterface  <class IOHIDInterface>
  |     +-o AppleHIDKeyboardEventDriverV2  <class AppleHIDKeyboardEventDriverV2>
  |       +-o IOHIDEventServiceUserClient  <class IOHIDEventServiceUserClient>
  +-o IOAccessoryIDBusBulkDataEndpoint@2  <class IOAccessoryIDBusBulkDataEndpoint>
    +-o AppleCS46L21IDBD  <class AppleCS46L21IDBD>
      +-o AppleCSI2CController  <class AppleCSI2CController>
        +-o 06ac-authcp  <class AppleARMIICDevice>
          +-o AppleAuthCPI2C  <class AppleAuthCPI2C>
```

The firmware updater is only interested in the instance of IOAccessoryIDBusHIDDevice that is handling the PrimaryUsage 11. On iOS devices it is possible for sandboxed code to talk to these HIDDevices and issue Set Feature Report and Get Feature Report commands. In the following sections we will discuss the most important reports.

5.1. Primary Usage 11: Get Feature Report

HID devices can return device specific information via feature reports. In order to do this the caller sends a Get Feature Report command to the opened HID device with one byte describing the number of the feature report we are interested in and the buffer size. The device then fills out the buffer (as needed) and returns the requested information. In the following list we will describe with examples the most important of these get feature reports as supported by the Apple Pencil and the Apple Smart Keyboard.

0x10 - Retrieve Device Information

Supported by: Apple Smart Keyboard

Returns a device information block from the device. verification result from last accessory firmware update process.

Offset	Length	Content
0x00	1 byte	Report Code 0x10
0x01	1 byte	Unknown - 0x02
0x02	2 byte	Vendor ID (0x5AC = Apple)
0x04	2 byte	Product ID
0x06	2 byte	Firmware Version
0x08	2 byte	Hardware Revision
0x0a	1 byte	Unknown - 0x00
0x0b	1 byte	Length of Serial
0x0c	x bytes	Serial Number

Example reply:

```
0x00: 10 02 ac 05 68 02 50 08 01 00 00 0c 44 51 44 51 ....h.P.....DQDQ
0x10: 50 41 54 36 47 57 54 4c                          PAT6GWTL
```

0x11 - Retrieve Serial Number

Supported by: Apple Smart Keyboard

Returns the device serial number.

Offset	Length	Content
0x00	1 byte	Report Code 0x11
0x01	x bytes	Serial Number

Example reply:

```
0x00: 11 44 51 44 51 50 41 54 36 47 57 54 4c          .DQDQPAT6GWTL
```

0x12 - Retrieve Interface Serial Number

Supported by: Apple Smart Keyboard

Returns a second device serial number that is called interface serial number.

Offset	Length	Content
0x00	1 byte	Report Code 0x12
0x01	x bytes	Interface Serial Number

Example reply:

```
0x00: 12 57 5a 53 35 34 30 34 30 47 4a 33 47 32 57 36 .WZS54040GJ3G2W6
0x10: 30 42                                           0B
```

0xB2 - Retrieve AFU Verification Result

Supported by: Apple Smart Keyboard / Apple Pencil

Returns the verification result from last accessory firmware update process.

Offset	Length	Content
0x00	1 byte	Report Code 0xB2
0x01	1 byte	Result (see appendix)

Example reply:

```
0x0: b2 b1                                            ..
```

0xB6 - Retrieve Data from SPI

Supported by: Apple Smart Keyboard 12.9" and 9.7"

Reads data via SPI from internal offset that can be set via set feature report 0xB6.

Offset	Length	Content
0x00	1 byte	Report Code 0xB6
0x01	3 bytes	Big Endian Offset
0x04	x bytes	Data from SPI

Example reply:

```
0x00: b6 00 00 00 ff ff ff ff ff ff ff ff ff ff ff ff ................
0x10: ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ................
0x20: ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ................
0x30: ff ff ff ff ff ff ff ff ff ff                   ..........
```

0xB8 - Retrieve AFU Protocol Information

Supported by: Apple Smart Keyboard / Apple Pencil

Returns information about the supported AFU protocol

Protocol 1

Offset	Length	Content
0x00	1 byte	Report Code 0xB8
0x01	2 byte	write buffer size

Example reply:

```
0x0: b8 30 00                                     .0.
```

0xDA - Retrieve Unique ID

Supported by: Apple Smart Keyboard

Returns the unique ID of the device. This is usually the 96 bit value of the STM32 Unique ID register.

Offset	Length	Content
0x00	1 byte	Report Code 0xDA
0x01	12 bytes	STM32 Unique ID

Example reply:

```
0x00: da 54 54 54 54 54 54 54 54 54 54 54          .TTTTTTTTTTT
```

5.2. Primary Usage 11: Set Feature Report

HID devices can react to device specific feature reports that allow controlling arbitrary aspects of a device. In order to do this the caller sends a Set Feature Report command to the opened HID device with the first byte being the number of the feature report we are interested in, followed by arbitrary command specific data. The device will then parse this request and process embedded information. In the following list we will describe with examples the most important of these set feature reports as supported by the Apple Pencil and the Apple Smart Keyboard.

0xB0 - Initialize AFU Transfer

Supported by: Apple Smart Keyboard / Apple Pencil

Initializes the AFU transfer.

Offset	Length	Content
0x00	1 byte	Report Code 0xB0
0x01	2 bytes	Code 0x6272 Init AFU Transfer, 0xC3BC Uber Init

Example report:

```
0x00: b0 72 62                                     .rb
```

0xB1 - Transfer AFU Data

Supported by: Apple Smart Keyboard / Apple Pencil

Sends an AFU data block to the accessory.

AFU Protocol with Offsets

Offset	Length	Content
0x00	1 byte	Report Code 0xB6
0x01	3 bytes	Data Offset
0x04	x bytes	Data to transfer

Example report:

```
0x00: b1 00 00 10 41 41 41                                  ....AAA
```

AFU Protocol without Offsets

Offset	Length	Content
0x00	1 byte	Report Code 0xB6
0x01	x bytes	Data to transfer

Example report:

```
0x00: b1 41 41 41                                           .AAA
```

0xB2 - Commit AFU Data

Supported by: Apple Smart Keyboard / Apple Pencil

Commits the transferred firmware.

Firmware Commit

Offset	Length	Content
0x00	1 byte	Report Code 0xB2

Example report:

```
0x00: b2                                                    .
```

Offset	Length	Content
0x00	1 byte	Report Code 0xB2
0x01	2 bytes	Commit Code (0xc3Bc - should come after commit without code)

Example report:

```
0x00: b2 bc c3                                          ...
```

0xB3 - Reset Device

Supported by: Apple Smart Keyboard / Apple Pencil

Resets the device.

Offset	Length	Content
0x00	1 byte	Report Code 0xB3

Example report:

```
0x00: b3                                                .
```

0xB6 - Set SPI Offset

Supported by: Apple Smart Keyboard

Allows to set the SPI offset for the next read operation.

Offset	Length	Content
0x00	1 byte	Report Code 0xB6
0x01	3 bytes	Big Endian Offset

Example report:

```
0x00: b6 00 00 10                                       ....
```

0xE1 - Control Crash Log

Supported by: Apple Pencil

Allows to control the Crash Log feature.

Subcommand 0x05: Set Crash Log Offset

Offset	Length	Content
0x00	1 byte	Report Code 0xE1
0x01	1 byte	0x05
0x02	3 bytes	Big Endian Crash Log Offset

Example report:

```
0x00: e1 05 00 00 10                                .....
```

Subcommand 0xCC: Create Test Crash Log Entry

Offset	Length	Content
0x00	1 byte	Report Code 0xE1
0x01	1 byte	0xCC

Example report:

```
0x00: e1 cc                                         ..
```

Subcommand 0xEE: Clear Crash Log

Offset	Length	Content
0x00	1 byte	Report Code 0xE1
0x01	1 byte	0xEE

Example report:

```
0x00: e1 ee                                         ..
```

Appendix - Asset Manifest Signing Cert

```
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:

                Full Name:
                  URI:http://www.apple.com/appleca/iphone.crl

            X509v3 Certificate Policies: critical
                Policy: 1.2.840.113635.100.5.8

    Signature Algorithm: sha1WithRSAEncryption
         9f:a0:a3:e1:78:e5:49:c4:48:87:5c:0f:17:c0:4b:0b:3f:dd:
         97:a0:ac:91:b4:ec:25:7c:f6:e2:d0:83:9b:78:1c:bd:bd:4e:
         bd:cf:ac:05:e8:35:27:2b:c7:8b:2d:82:e0:dc:44:18:36:2c:
         55:81:ef:51:bb:9c:5e:b1:9d:ba:5e:a4:29:ab:8e:61:c1:33:
         a6:88:24:e7:a0:48:45:e9:ca:1e:2b:c0:a9:04:2d:34:9a:55:
         9a:aa:c4:96:0a:83:9e:d2:3f:6b:3d:86:b3:0d:f6:4c:b5:b2:
         0d:75:70:f2:b5:a2:bd:c8:59:13:e5:4e:bf:59:93:e1:ae:cb:
         1e:c7:71:31:66:c4:38:8a:e4:be:46:54:3a:8c:59:95:08:63:
         3c:27:36:da:6a:74:63:2e:6b:c1:c7:fe:df:f2:30:24:63:a1:
         25:25:ce:56:b7:31:b8:ec:26:c1:e3:6b:2b:51:1a:8e:7f:87:
         4b:62:25:9c:c0:70:40:39:07:fd:de:f8:29:45:c6:be:e3:57:
         d0:bb:17:bf:d7:02:40:c0:7c:b5:c8:c5:96:bd:6e:f9:2c:e8:
         5c:21:c3:d5:22:64:53:87:b1:dd:ab:a1:9e:e9:18:4a:88:9a:
         83:bf:2d:c2:60:d1:93:bd:ba:69:89:b1:34:87:d8:62:90:ec:
         d2:14:bb:33
```

-----BEGIN CERTIFICATE-----
MIID7TCCAtWgAwIBAgIBLTANBgkqhkiG9w0BAQUFADB5MQswCQYDVQQGEwJVUzET
MBEGA1UEChMKQXBwbGUgSW5jLjEmMCQGA1UECxMdQXBwbGUgQ2VydGlmaWNhdGlv
biBBdXRob3JpdHkxLTArBgNVBAMTJEFwcGxlIGlQaG9uZSBDZXJ0aWZpY2F0aW9u
IEF1dGhvcml0eTAeFw0xMTA3MTQyMjMyNDhaFw0xODA3MTQyMjMyNDhaMGYxCzAJ
BgNVBAYTAlVTMRMwEQYDVQQKEwpBcHBsZSBJbmMuMSEwHwYDVQQLExhBcHBsZSBp
T1MgQXNzZXQgTWFuaWZlc3QxHzAdBgNVBAMTFkFzc2V0IE1hbmlmZXN0IFNpZ25p
bmcwggEiMA0GCSqGSIb3DQEBAQUAA4IBDwAwggEKAoIBAQC1y4D4VYBPHLwnHk1q
fi1F49E3Ml675ElYYAFghqpAiKOqeagLVZU8zKuPw650NHACvNJbjKBjMsWdWaFM
j/7c2TB5IkJwi61Y6B+uVK78W9u9I/hFAK0pWcM9Y5KbKMvz4wEwt64EXfS8eVBR
mqh/29ygxN9NtBbHEiGiGQ8vxIV3U6FomNdmxKPM7VZmsyFIxQ5HsxgHb0skxlDI
dePtYsHLmpK9PX43K3sBT3lHN0Uxtit8HTrdwiNq13cI0TINT+lsbXKLp3/gPJVp
fxkQ3MTt40KG/DTMsqKOygDombsFSqA+RPSv68YtJ/kAaGaoH6gZfRrwX7GJozzQ
8KTZAgMBAAGjgZIwgY8wDgYDVR0PAQH/BAQDAgeAMAkGA1UdEwQCMAAwHQYDVR0O
BBYEFGNQxP6y1Ao4Hrhid6BcMLwcrB3BMDgGA1UdHwQxMC8wLaAroCmGJ2h0dHA6
Ly93d3cuYXBwbGUuY29tL2FwcGxlY2EvaXBob25lLmNybDAZBgNVHSABAf8EDzAN
MAsGCSqGSIb3Y2QFCDANBgkqhkiG9w0BAQUFAAOCAQEAn6Cj4XjlScRIh1wPF8BL
Cz/dl6CskbTsJXz24tCDm3gcvb1Ovc+sBeg1JyvHiy2C4NxEGDYsVYHvUbucXrGd
ul6kKauOYcEzpogk56BIRenKHivAqQQtNJpVmqrElgqDntI/az2Gsw32TLWyDXVw
8rWivchZE+VOv1mT4a7LHsdxMWbEOIrkvkZUOoxZlQhjPCc22mp0Yy5rwcf+3/Iw
JGOhJSXOVrcxuOwmweNrK1Eajn+HS2IlnMBwQDkH/d74KUXGvuNX0LsXv9cCQMB8
tcjFlr1u+SzoXCHD1SJkU4ex3auhnukYSoiag78twmDRk726aYmxNIfYYpDs0hS7
Mw==
-----END CERTIFICATE-----

Appendix - AFU Result Codes

Code (hex)	Code (dec)	Constant
0x00	0	kAUErrorNone
0x01	1	kAUErrorAccessoryBusy
0x02	2	kAUErrorReadFailure
0x03	3	kAUErrorStoreFailure
0x04	4	kAUErrorEraseFailure
0x10	16	kAUErrorHdrSigInvalid
0x11	17	kAUErrorHdrVerUnsupported
0x12	18	kAUErrorHdrCRCMismatch
0x13	19	kAUErrorPayloadTypeInvalid
0x14	20	kAUErrorPayloadVerDowngrade
0x15	21	kAUErrorPayloadSizeInvalid
0x16	22	kAUErrorPayloadCRCMismatch
0x17	23	kAUErrorPayloadPIDMismatch
0x18	24	kAUErrorPayloadHWIDMismatch
0x40	64	kAUErrorInvalidRegion
0x70	112	kAUErrorSigInvalid
0x71	113	kAUErrorSigDigestMismatch
0x72	114	kAUErrorSigInfoInvalid
0x73	115	kAUErrorSigBadMagic
0x74	116	kAUErrorSigVerUnsupported
0x75	117	kAUErrorSigBadDigestInfo
0x76	118	kAUErrorSigBadSigInfo
0x77	119	kAUErrorNotSigned
0x80	128	kAUErrorNotPersonalized
0x81	129	kAUErrorPersVerUnsupported
0x82	130	kAUErrorPersUDIDMismatch
0x83	131	kAUErrorPersBadMagic
0xA1	161	kAUErrorSuccessKey
0xB0	176	kAUErrorInitKeyIncorrect
0xB1	177	kAUErrorInvalidOperation
0xFF	255	kAUErrorUnknown