USER’S GUIDE

Apollo3 Blue Secure Bootloader Scripts

Ultra-Low Power Apollo SoC Family

A-SOCA3B-UGGA01EN v3.1

Legal Information and Disclaimers

AMBIQ MICRO INTENDS FOR THE CONTENT CONTAINED IN THE DOCUMENT TO BE ACCURATE AND RELIABLE. THIS CONTENT MAY, HOWEVER, CONTAIN TECHNICAL INACCURACIES, TYPOGRAPHICAL ERRORS OR OTHER MISTAKES. AMBIQ MICRO MAY MAKE CORRECTIONS OR OTHER CHANGES TO THIS CONTENT AT ANY TIME. AMBIQ MICRO AND ITS SUPPLIERS RESERVE THE RIGHT TO MAKE CORRECTIONS, MODIFICATIONS, ENHANCEMENTS, IMPROVEMENTS AND OTHER CHANGES TO ITS PRODUCTS, PROGRAMS AND SERVICES AT ANY TIME OR TO DISCONTINUE ANY PRODUCTS, PROGRAMS, OR SERVICES WITHOUT NOTICE.

THE CONTENT IN THIS DOCUMENT IS PROVIDED "AS IS". AMBIQ MICRO AND ITS RESPECTIVE SUPPLIERS MAKE NO REPRESENTATIONS ABOUT THE SUITABILITY OF THIS CONTENT FOR ANY PURPOSE AND DISCLAIM ALL WARRANTIES AND CONDITIONS WITH REGARD TO THIS CONTENT, INCLUDING BUT NOT LIMITED TO, ALL IMPLIED WARRANTIES AND CONDITIONS OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT OF ANY THIRD PARTY INTELLECTUAL PROPERTY RIGHT.

AMBIQ MICRO DOES NOT WARRANT OR REPRESENT THAT ANY LICENSE, EITHER EXPRESS OR IMPLIED, IS GRANTED UNDER ANY PATENT RIGHT, COPYRIGHT, MASK WORK RIGHT, OR OTHER INTELLECTUAL PROPERTY RIGHT OF AMBIQ MICRO COVERING OR RELATING TO THIS CONTENT OR ANY COMBINATION, MACHINE, OR PROCESS TO WHICH THIS CONTENT RELATE OR WITH WHICH THIS CONTENT MAY BE USED.

USE OF THE INFORMATION IN THIS DOCUMENT MAY REQUIRE A LICENSE FROM A THIRD PARTY UNDER THE PATENTS OR OTHER INTELLECTUAL PROPERTY OF THAT THIRD PARTY, OR A LICENSE FROM AMBIQ MICRO UNDER THE PATENTS OR OTHER INTELLECTUAL PROPERTY OF AMBIQ MICRO.

INFORMATION IN THIS DOCUMENT IS PROVIDED SOLELY TO ENABLE SYSTEM AND SOFTWARE IMPLEMENTERS TO USE AMBIQ MICRO PRODUCTS. THERE ARE NO EXPRESS OR IMPLIED COPYRIGHT LICENSES GRANTED HEREUNDER TO DESIGN OR FABRICATE ANY INTEGRATED CIRCUITS OR INTEGRATED CIRCUITS BASED ON THE INFORMATION IN THIS DOCUMENT. AMBIQ MICRO RESERVES THE RIGHT TO MAKE CHANGES WITHOUT FURTHER NOTICE TO ANY PRODUCTS HEREIN. AMBIQ MICRO MAKES NO WARRANTY, REPRESENTATION OR GUARANTEE REGARDING THE SUITABILITY OF ITS PRODUCTS FOR ANY PARTICULAR PURPOSE, NOR DOES AMBIQ MICRO ASSUME ANY LIABILITY ARISING OUT OF THE APPLICATION OR USE OF ANY PRODUCT OR CIRCUIT, AND SPECIFICALLY DISCLAIMS ANY AND ALL LIABILITY, INCLUDING WITHOUT LIMITATION CONSEQUENTIAL OR INCIDENTAL DAMAGES. “TYPICAL” PARAMETERS WHICH MAY BE PROVIDED IN AMBIQ MICRO DATA SHEETS AND/OR SPECIFICATIONS CAN AND DO VARY IN DIFFERENT APPLICATIONS AND ACTUAL PERFORMANCE MAY VARY OVER TIME. ALL OPERATING PARAMETERS, INCLUDING “TYPICALS” MUST BE VALIDATED FOR EACH CUSTOMER APPLICATION BY CUSTOMER’S TECHNICAL EXPERTS. AMBIQ MICRO DOES NOT CONVEY ANY LICENSE UNDER NEITHER ITS PATENT RIGHTS NOR THE RIGHTS OF OTHERS. AMBIQ MICRO PRODUCTS ARE NOT DESIGNED, INTENDED, OR AUTHORIZED FOR USE AS COMPONENTS IN SYSTEMS INTENDED FOR SURGICAL IMPLANT INTO THE BODY, OR OTHER APPLICATIONS INTENDED TO SUPPORT OR SUSTAIN LIFE, OR FOR ANY OTHER APPLICATION IN WHICH THE FAILURE OF THE AMBIQ MICRO PRODUCT COULD CREATE A SITUATION WHERE PERSONAL INJURY OR DEATH MAY OCCUR. SHOULD BUYER PURCHASE OR USE AMBIQ MICRO PRODUCTS FOR ANY SUCH UNINTENDED OR UNAUTHORIZED APPLICATION, BUYER SHALL INDEMNIFY AND HOLD AMBIQ MICRO AND ITS OFFICERS, EMPLOYEES, SUBSIDIARIES, AFFILIATES, AND DISTRIBUTORS HARMLESS AGAINST ALL CLAIMS, COSTS, DAMAGES, AND EXPENSES, AND REASONABLE ATTORNEY FEES ARISING OUT OF, DIRECTLY OR INDIRECTLY, ANY CLAIM OF PERSONAL INJURY OR DEATH ASSOCIATED WITH SUCH UNINTENDED OR UNAUTHORIZED USE, EVEN IF SUCH CLAIM ALLEGES THAT AMBIQ MICRO WAS NEGLIGENT REGARDING THE DESIGN OR MANUFACTURE OF THE PART.

Revision History

Revision	Date	Description
0.1	March 22, 2018	Initial version
0.2	March 29, 2018	Updated details and created separate sections for Non-Secure and Secure. Also separated out external content
1.0	March 29, 2018	Apollo3 Blue SDK Alpha Release
2.0		SBLv1 Release
2.1	July 5, 2018	SBLv2 Release Also added an example for multi-image upgrade using UART script. Added sections for SBL and Patch upgrade
2.2	September 24, 2019	Changes for Apollo3 Blue Plus
2.3	November 8, 2019	Added 8.1.8 for --split parameter
3.0	December 16, 2021	Updates to create_info0. Also updated parameters to match the EVB.
3.1	October 19, 2022	Template update

Reference Documents

Document ID	Description
A-SOCA3B-UGGA02EN	Apollo3 Blue Secure Update Flow
A-SOCAP3-UGGA04EN	AMOTA Example User's Guide

Table of Contents

List of Figures

SECTION

1

Introduction

Ambiq Apollo3 Blue SDK contains a number of python scripts to demonstrate generation of Customer InfoSpace (INFO0) settings, Customer Main images, and creation of images for the Wired Update protocol over UART. This document will explain their usage.

These scripts have been upgraded to be compatible with Apollo3 Blue Plus, with the following changes:

- Extension of the Permanent Write Protections, Permanent Copy Protections, and SBL Overridable Write Protections, and SBL Overridable Copy Protections for the new Flash Instances #2 and #3.
- New Chip Type parameter to distinguish between Apollo3 Blue and Apollo3 Blue Plus.

SECTION

2

Preparation of the Python Environment

This document assumes that the user has a python3 environment available. The SBL scripts require the addition of the python crypto modules. Those can be obtained as follows:

```
pip install pycryptodome
pip install pyserial
```

Most of the python scripts discussed in this document can be found in **/tools/apollo3_scripts/**

OTA related scripts are placed in **/tools/amota/scripts/**

SECTION

3

Keys

Most of the Python scripts expect a file named **keys_info.py** to be present in the same directory.

This file contains all the sensitive key information – which are either controlled by the customers themselves, or obtained through Ambiq.

These keys are used to generate InfoSpace images, and to generate encrypted/signed images and required for encrypting/signing the wired update messages, as per customer requirements.

Ambiq SDK provides a template file **keys_info0.py** with dummy values, which is to be edited by the customer with correct values and renamed as **keys_info.py**.

This file contains definition of:

- **keyTblAes**: Infospace Decryption Keys
- **keyTblHmac**: Infospace Authentication Keys
- **custKey**: 128b Customer defined Security Key which protects Infospace Read Access
- **recoveryKey:** 128b Unique Key value provided by Ambiq – used for device recovery

SECTION

4

Image Generation Scripts

4.1 Generating Customer InfoSpace (INFO0)

INFO0 space on the target is 8K of separate flash area, which dictates the device behavior in a number of ways.

Script **create_info0.py** can be used to create a binary file to be populated as INFO0. It uses the key information in **keys_info.py** and allows the user to define a number of other INFO0 parameters based on command line.

```
usage: create_info0.py [-h] [--valid {0,1,2}] [--version VERSION]
                       [--main MAINPTR] [--secpol {0,1,2,3,4,5,6,7}]
                       [--wrap {0,1,2}] [--sRst {0,1}] [-s {0,1}] [--pl {0,1}]
                       [--sDbgAllowed {0,1}] [--erase {0,1}] [--prog INFOPROG]
                       [--snowipe {0,1}] [--swo {0,1}] [--dbgprot {0,1}]
                       [--trim CUSTTRIM] [--trim2 CUSTTRIM2]
                       [--gpio OVERRIDEGPIO] [--gpiolvl {0,1}]
                       [--wmask WIREDIFMASK] [--wSlInt WIREDSLVINT]
                       [--wI2c WIREDI2CADDR] [--wTO WIREDTIMEOUT] [--u0 U0]
                       [--u1 U1] [--u2 U2] [--u3 U3] [--u4 U4] [--u5 U5]
                       [--krev KREV] [--arev AREV] [--sresv SRESV]
                       [--chipid0 CHIPID0] [--chipid1 CHIPID1]
                       [--wprot0 WPROT0] [--wprot1 WPROT1] [--rprot0 RPROT0]
                       [--rprot1 RPROT1] [--swprot0 SWPROT0]
                       [--swprot1 SWPROT1] [--srprot0 SRPROT0]
                       [--srprot1 SRPROT1] [--wprot2 WPROT2] [--wprot3 WPROT3]
                       [--rprot2 RPROT2] [--rprot3 RPROT3] [--swprot2 SWPROT2]
                       [--swprot3 SWPROT3] [--srprot2 SRPROT2]
                       [--srprot3 SRPROT3] [--chipType {apollo3,apollo3p}]
                       [-k [KEYFILE]] [--loglevel {0,1,2,3,4,5}]
                       output
```

```
Generate Corvette Info0 Blob

positional arguments:
  output                Output filename (without the extension)

optional arguments:
  -h, --help            show this help message and exit
  --valid {0,1,2}       INFO0 Valid 0 = Uninitialized, 1 = Valid, 2 = Invalid
                        (Default = 1)?
  --version VERSION     version (Default = 0)?
  --main MAINPTR        Main Firmware location (Default = 0xc000)?
  --secpol {0,1,2,3,4,5,6,7}
                        Security Policy Bitmask (Default = 0)? (bit 0 = Auth,
                        bit 1 = Enc, bit 2 = Version Rollback)
  --wrap {0,1,2}        KeyWrap Algo (Default = 0)? (0 = none, 1 = XOR, 2 =
                        AES128)
  --sRst {0,1}          Secure Boot on Soft Reset (Default = 0) ?
  -s {0,1}              Secure Boot (Default = 0) ?
  --pl {0,1}            Protection Lock Enabled (Default = 0) ?
  --sDbgAllowed {0,1}   Debugger allowed during (optional) Secondary
                        Bootloader (Default = 1) ?
  --erase {0,1}         Info0 Erase Allowed (Default = 1) ?
  --prog INFOPROG       INFO0 Program allowed (1 bit per quadrant) (Default =
                        0xf) ?
  --snowipe {0,1}       Do not wipe SRAM on debugger connection (Default = 1)
                        ?
  --swo {0,1}           debugger connection allowed (Default = 1) ?
  --dbgprot {0,1}       Do not lock debugger (Default = 1) ?
  --trim CUSTTRIM       customer trim ?
  --trim2 CUSTTRIM2     customer trim 2?
  --gpio OVERRIDEGPIO   Override GPIO (7 bit - in hex) - 0x7f for disabled
                        (Default = 0x7f)
  --gpiolvl {0,1}       Override GPIO Polarity (0 = low, 1 = hi) (Default = 0)
  --wmask WIREDIFMASK   Wired interface mask (bit 0 = UART, bit 1 = SPI, bit 2
                        = I2C) (default = UART)
  --wSlInt WIREDSLVINT  Wired IOS interface handshake pin (default = 4)
  --wI2c WIREDI2CADDR   Wired IOS interface I2C Address (default = 0x20)
  --wTO WIREDTIMEOUT    Wired interface timeout in millisec (default = 20000)
  --u0 U0               UART Config 0 (default = 0xFFFFFFFF)
  --u1 U1               UART Config 1 (default = 0xFFFFFFFF)
  --u2 U2               UART Config 2 (default = 0xFFFFFFFF)
  --u3 U3               UART Config 3 (default = 0xFFFFFFFF)
  --u4 U4               UART Config 4 (default = 0xFFFFFFFF)
  --u5 U5               UART Config 5 (default = 0xFFFFFFFF)
  --krev KREV           KEK Revocation Mask (Default 0xFFFFFFFF)
  --arev AREV           AuthKey Revocation Mask (Default 0xFFFFFFFF)
  --sresv SRESV         SRAM Reservation (Default 0x0)
  --chipid0 CHIPID0     CHIPID0 for the device (Default 0)
  --chipid1 CHIPID1     CHIPID1 for the device (Default 0)
  --wprot0 WPROT0       Permanent Write Protections Mask for flash#0 (Default
                        0xFFFFFFFF)
  --wprot1 WPROT1       Permanent Write Protections Mask for flash#1 (Default
                        0xFFFFFFFF)
  --rprot0 RPROT0       Permanent Copy Protections Mask for flash#0 (Default
                        0xFFFFFFFF)
  --rprot1 RPROT1       Permanent Copy Protections Mask for flash#1 (Default
                        0xFFFFFFFF)
  --swprot0 SWPROT0     SBL overridable Write Protections Mask for flash#0
                        (Default 0xFFFFFFFF)
  --swprot1 SWPROT1     SBL overridable Write Protections Mask for flash#1
                        (Default 0xFFFFFFFF)
  --srprot0 SRPROT0     SBL overridable Copy Protections Mask for flash#0
                        (Default 0xFFFFFFFF)
  --srprot1 SRPROT1     SBL overridable Copy Protections Mask for flash#1
                        (Default 0xFFFFFFFF)
  --wprot2 WPROT2       Permanent Write Protections Mask for flash#2 (Default
                        0xFFFFFFFF)
  --wprot3 WPROT3       Permanent Write Protections Mask for flash#3 (Default
                        0xFFFFFFFF)
  --rprot2 RPROT2       Permanent Copy Protections Mask for flash#2 (Default
                        0xFFFFFFFF)
  --rprot3 RPROT3       Permanent Copy Protections Mask for flash#3 (Default
                        0xFFFFFFFF)
  --swprot2 SWPROT2     SBL overridable Write Protections Mask for flash#2
                        (Default 0xFFFFFFFF)
  --swprot3 SWPROT3     SBL overridable Write Protections Mask for flash#3
                        (Default 0xFFFFFFFF)
  --srprot2 SRPROT2     SBL overridable Copy Protections Mask for flash#2
                        (Default 0xFFFFFFFF)
  --srprot3 SRPROT3     SBL overridable Copy Protections Mask for flash#3
                        (Default 0xFFFFFFFF)
  --custId CUSTID       CUSTID (Default 0xFFFFFFFF)
  --chipType {apollo3,apollo3p}
                        Chip Type: apollo3, apollo3p (default = apollo3)
  -k [KEYFILE]          key file in specified format [default = keys_info.py]
  --loglevel {0,1,2,3,4,5}
                        Set Log Level (0: None), (1: Error), (2: INFO), (4:
                        Verbose), (5: Debug) [Default = Info]
```

4.1.1 Example Usage

4.1.1.1 Create INFO0 Image for Non-Secure Usage

Create INFO0 image with GPIO Override is set to pin 16 (0x10) active low. Baudrate for INFO0 UART is set to 115200 (0x1C200). Main image is expected at 0xC000. Apollo3* is configured for UART-RX pin 23 (0x17) and UART-TX pin 22 (0x16). The chip type is specified as apollo3p/apollo3.

```
./create_info0.py --valid 1 info0 --pl 1 --u0 0x1C200c0 --u1 0xFFFF1617 -
-u2 0x2 --u3 0x0 --u4 0x0 --u5 0x0 --main 0xC000 --gpio 0x10 --version 0
--wTO 5000 --chipType apollo3p
```

Optionally, Ambiq provided unique Customer ID can be programmed as well (relevant only if transitioning to Secure SKU later).

```
./create_info0.py --valid 1 info0 --pl 1 --u0 0x1C200c0 --u1 0xFFFF1617 -
-u2 0x2 --u3 0x0 --u4 0x0 --u5 0x0 --main 0xC000 --gpio 0x10 --version 0
--wTO 5000 --custID 0xDEADBEEF --chipType apollo3px
```

4.1.1.2 Create INFO0 Image for secure Usage (Only applicable for Secure SKU)

Create INFO0 image with GPIO Override is set to pin 16 (0x10) active low. Baudrate for INFO0 UART is set to 115200 (0x1C200). Main image is expected at 0xC000. Apollo3* is configured for UART-RX pin 23 (0x17) and UART-TX pin 22 (0x16). The chip type is specified as apollo3p/apollo3. A valid (Ambiq Supplied) CustID is programmed and Secure Boot is enabled.

```
./create_info0.py --valid 1 info0 --pl 1 --u0 0x1C200c0 --u1 0xFFFF1617 -
-u2 0x2 --u3 0x0 --u4 0x0 --u5 0x0 --main 0xC000 --gpio 0x10 --version 0
--wTO 5000 –s 1 --custId 0xDEADBEEF --chipType apollo3p
```

4.2 Generating Customer Firmware Images

Apollo3 Blue and Apollo3 Blue Plus SBL recognizes a number of different image types.

- Main (Secure Firmware)
- NonSecure (Non-Secure Firmware)
- Child (3rd Party firmware libraries)
- Info0 (Info0 Update Binary)
- CustOTA (Other) – Used with Secondary bootloader to pass through customer specific upgrade image types

Details of individual image formats is described in a separate document (*Apollo3 Blue Secure Update Flow User's Guide*).

Script **create_cust_image_blob.py** can be used to create a binary image blob as understood by the SBL.

The images generated such are good to be used directly with Flash Programming Tools (Jflash/ JFlashLite, IAR, Keil), or transferred to the device wirelessly using customer defined OTA protocol and application.

```
usage: create_cust_image_blob.py [-h] [--bin APPFILE]
                                 [--load-address LOADADDRESS]
                                 [--magic-num {0xc0,0xcc,0xc1,0xcb,0xcf}]
                                 [-o OUTPUT]
                                 [--authkey {8,9,10,11,12,13,14,15}]
                                 [--kek {8,9,10,11,12,13,14,15}]
                                 [--authalgo {0,1}] [--encalgo {0,1}]
                                 [--child0 CHILD0] [--child1 CHILD1]
                                 [--version VERSION] [--crcI {0,1}]
                                 [--crcB {0,1}] [--authI {0,1}]
                                 [--authB {0,1}] [--erasePrev {0,1}]
                                 [-p {0,1,2,3}] [-k [KEYFILE]]
                                 [--loglevel {0,1,2,3,4,5}]

Generate Corvette Image Blob

optional arguments:
  -h, --help            show this help message and exit
  --bin APPFILE         binary file (blah.bin)
  --load-address LOADADDRESS
                        Load address of the binary.
  --magic-num {0xc0,0xcc,0xc1,0xcb,0xcf}
                        Magic Num (0xc0: Main, 0xcc: Child, 0xc1: CustOTA,
                        0xcb: NonSecure, 0xcf: Info0) - default[Main]
  -o OUTPUT             Output filename (without the extension)
  --authkey {8,9,10,11,12,13,14,15}
                        Authentication Key Idx? (8 to 15)
  --kek {8,9,10,11,12,13,14,15}
                        KEK Index? (8 to 15)
  --authalgo {0,1}      Authentication Algo? (0(default) = none, 1 = SHA256)
  --encalgo {0,1}       Encryption Algo? (0(default) = none, 1 = AES128)
  --child0 CHILD0       child (blobPtr#0 for Main / feature key for AM3P)
  --child1 CHILD1       child (blobPtr#1 for Main)
  --version VERSION     version (15 bit)
  --crcI {0,1}          Install CRC check enabled (Default = Y)?
  --crcB {0,1}          Boot CRC check enabled (Default = N)?
  --authI {0,1}         Install Authentication check enabled (Default = N)?
  --authB {0,1}         Boot Authentication check enabled (Default = N)?
  --erasePrev {0,1}     erasePrev (Valid only for main)
  -p {0,1,2,3}          protection info 2 bit C W
  -k [KEYFILE]          key file in specified format [default = keys_info.py]
  --loglevel {0,1,2,3,4,5}
                        Set Log Level (0: None), (1: Error), (2: INFO), (4:
                        Verbose), (5: Debug) [Default = Info]
```

4.2.1 Example Usage

4.2.1.1 Create a Non-Secure Customer Image

Create a non-secure customer image from a built binary with Flash base address of 0xC000 (hello_world.bin). This is the Customer Main Non-Secure format from the *Apollo3 Blue Secure Update Flow User's Guide*.

```
./create_cust_image_blob.py --bin hello_world.bin --load-address 0xC000
--magic-num 0xCB -o main_nonsecure_ota --version 0x0
```

4.2.1.2 Create a Secure Customer Image

Create a secure customer image from a built binary with Flash base address of 0xC100 (hello_world_0xc100.bin). This is the Customer Main format from the *Apollo3 Blue Secure Update Flow User's Guide.* AES128 encryption (kek, encalgo) and SHA256-HMAC based authentication is enabled (authkey, authalgo, authI). Post install boots should verify the signature for authenticity (authB).

```
./create_cust_image_blob.py --bin hello_world_0xc100.bin --load-address
0xC000 --magic-num 0xC0 -o main_secure_ota --version 0x0 --kek 8 --auth-
key 10 --encalgo 1 --authalgo 1 --authB 1 --authI 1
```

SECTION

5

Generating Wired Update Images

To facilitate Wired update using SBL through an external host, the image blobs are further encapsulated in a predefined format as described in *Apollo3 Blue Secure Update Flow User's Guide*.

Script **create_cust_wireupdate_blob.py** facilitates generation of these encapsulated images. This script also internally takes care of generating split encapsulated images, if the image size is bigger than what can be accepted by SBL in one transaction over wired interface.

```
usage: create_cust_wireupdate_blob.py [-h] [--load-address LOADADDRESS]
                                      [--bin APPFILE]
                                      [-i {0,1,2,3,4,5,6,7,32}]
                                      [--options OPTIONS] [-o OUTPUT]
                                      [--authkey {8,9,10,11,12,13,14,15}]
                                      [--kek {8,9,10,11,12,13,14,15}]
                                      [--authalgo {0,1}] [--encalgo {0,1}]
                                      [--split SPLIT] [-k [KEYFILE]]
                                      [--loglevel {0,1,2,3,4,5}]

Generate Corvette Wired Update Blob

optional arguments:
  -h, --help            show this help message and exit
  --load-address LOADADDRESS
                        Load address of the binary - Where in flash the blob
                        will be stored (could be different than install
                        address of binary within).
  --bin APPFILE         binary file (blah.bin)
  -i {0,1,2,3,4,5,6,7,32}
                        ImageType (0: SBL, 1: AM3P, 2: Patch, 3: Main, 4:
                        Child, 5: CustOTA, 6: NonSecure, 7: Info0, 32:
                        Info0-NoOTA) - default[Main]
  --options OPTIONS     Options (16b hex value) - bit0 instructs to perform
                        OTA of the image after wired download (set to 0 if
                        only downloading and skipping OTA flow)
  -o OUTPUT             Output filename (without the extension)
  --authkey {8,9,10,11,12,13,14,15}
                        Authentication Key Idx? (8 to 15)
  --kek {8,9,10,11,12,13,14,15}
                        KEK Index? (8 to 15)
  --authalgo {0,1}      Authentication Algo? (0(default) = none, 1 = SHA256)
  --encalgo {0,1}       Encryption Algo? (0(default) = none, 1 = AES128)
  --split SPLIT         Specify the max block size if the image will be
                        downloaded in pieces
  -k [KEYFILE]          key file in specified format [default = keys_info.py]
  --loglevel {0,1,2,3,4,5}
                        Set Log Level (0: None), (1: Error), (2: INFO), (4:
                        Verbose), (5: Debug) [Default = Info]
```

5.1 Example Usage

5.1.1 Create Non-Secure Wired Update Image Blob

Create Non-Secure Wired Update Image blob corresponding to the Upgrade image (generated as in *Section 4.2.1 Example Usage on page 15*), as shown in the *Apollo3 Blue Secure Update Flow User's Guide*:

```
./create_cust_wireupdate_blob.py --load-address 0x20000 --bin main_non-
secure_ota.bin -i 6 -o main_nonsecure_wire --options 0x1
```

5.1.2 Create INFO0-NOOTA Wired Update Image Blob

Create INFO0-NOOTA Wired Update Image blob from the INFO0 image (generated as in *Section 4.1.1 Example Usage on page 13*) in the previous step:

```
./create_cust_wireupdate_blob.py --bin info0.bin -o info0_wire -i 32 --
load-address 0
```

5.1.3 Create Secure Bootloader (SBL) Wired Update Image Blob

Create SBL Wired Update Image blob corresponding to the Upgrade image (provided by Ambiq), as shown in the *Apollo3 Blue Secure Update Flow User's Guide*:

```
./create_cust_wireupdate_blob.py --load-address 0x20000 --bin
sbl_ota.bin -i 0 -o sbl_wire --options 0x1
```

5.1.4 Create Secure Wired Update Image Blob

Create Secure Wired Update Image blob corresponding to the Secure or non-secure Upgrade image (generated as in *Section 4.2.1 Example Usage on page 15*), as shown in the *Apollo3 Blue Secure Update Flow User's Guide*:

```
./create_cust_wireupdate_blob.py --load-address 0x20000 --bin main_non-
secure_ota.bin -i 6 -o main_nonsecure_swire --options 0x1  --kek 11 --
authkey 10 --encalgo 1 --authalgo 1
```

OR (for secure main image)

```
./create_cust_wireupdate_blob.py --load-address 0x20000 --bin main_se-
cure_ota.bin -i 3 -o main_secure_swire --options 0x1  --kek 11 --authkey
10 --encalgo 1 --authalgo 1
```

Secure wired update ensures that only trusted host is allowed to download any upgrades to the device. Note that this is not to be confused with the nonsecure or secure image itself, which pertains to validation enforcements by the SBL on the installed images.

5.1.5 Create Patch Wired Update Image Blob

Create Patch Wired Update Image blob corresponding to the Upgrade image (provided by Ambiq), as shown in the *Apollo3 Blue Secure Update Flow User's Guide*:

```
./create_cust_wireupdate_blob.py --load-address 0x20000 --bin
patch_ota.bin -i 2 -o patch_wire --options 0x1
```

SECTION 6

Creating Device Recovery Message

A corrupt INFO0 (e.g., invalid signature, or invalid values for Security fields) on a Secure SKU Apollo3-Blue SoC causes SBL to go into a “recovery” mode. The only option possible in this case is to use the Wired Update feature to send a “RECOVER” message with proper credentials to do a factory-reset.

To generate a RECOVER message with proper credentials, customer would need to contact Ambiq using a secure channel and provide certain details (Unique CustomerID assigned to them, a range of CHIP Part#s along with a unique 128b Nonce value). Ambiq will then provide an “Ambiq Recovery Blob”, which is bound to the CustomerID, Nonce and particular part.

Even for non-secure SKU’s this procedure can be used to revert back to factory settings.

Script **create_recover_message.py** can then be used to generate the “RECOVER” message using the Ambiq Recovery Blob, along with customer supplied Nonce andCustomerID.

```
usage: create_recover_msg.py [-h] [-f BINFILE] [-o OUTPUT] [--n0 N0] [--n1 N1]
                             [--n2 N2] [--n3 N3] [--custId CUSTID]
Generate Corvette Recovery Message
optional arguments:
  -h, --help       show this help message and exit
  -f BINFILE       Binary file representing the raw Recovery Blob provided by
                   Ambiq
  -o OUTPUT        Output filename (without the extension)
  --n0 N0          Nonce 0 - should correspond to the value provided to Ambiq
                   (default = 0xFFFFFFFF)
  --n1 N1          Nonce 1 - should correspond to the value provided to Ambiq
                   (default = 0xFFFFFFFF)
  --n2 N2          Nonce 2 - should correspond to the value provided to Ambiq
                   (default = 0xFFFFFFFF)
  --n3 N3          Nonce 3 - should correspond to the value provided to Ambiq
                   (default = 0xFFFFFFFF)
  --custId CUSTID  Customer ID - should correspond to the value provided to
                   Ambiq (default = 0xFFFFFFFF)
```

6.1 Example Usage

6.1.1 Create Secure recover message

Create Secure recover message for customer with custID 0x1000, with supplied Ambiq Recover Blob in am_rec.bin (generated corresponding to nonce 0x0, 0xDEADBEEF, 0xFFFFFFFF, 0xA5A55A5A)

```
./create_recover_msg.py -f am_rec.bin -o recover_secure --n0 0 --n1
0xDEADBEEF --n2 0xFFFFFFFF --n3 0xA5A55A5A --custId 0x1000
```

6.1.2 Create Non-Secure recover message

Create Non-Secure recover message for customer with custID 0x1000 (generated corresponding to nonce 0x0, 0xDEADBEEF, 0xFFFFFFFF, 0xA5A55A5A)

```
./create_recover_msg.py -o recover
```

SECTION

7

UART Wired Update

For UART based wired update to work, the device needs to be provisioned to allow UART wired update through InfoSpace settings. SBL will get into update mode in one of the two cases:

- Encountering fatal error (e.g., invalid main image)
- GPIO Override (configured through InfoSpace)

The host needs to be connected to the device on the configured pins to match with the InfoSpace UART configurations, and needs to initiate the communication within a short window configured (through InfoSpace).

Script **uart_wired_update.py** is designed to emulate the host side functions in a limited way when using the UART as wired interface.

```
usage: uart_wired_update.py [-h] [-b BAUD] [--raw RAW] [-f BINFILE]
                            [-i {0,1,2,3,4,5,6,7,32,255}] [-o OTADESC]
                            [-r {0,1,2}] [-a {0,1,-1}] [--split SPLIT]
                            port
UART Wired Update Host for Apollo3
positional arguments:
  port                  Serial COMx Port

optional arguments:
  -h, --help            show this help message and exit
  -b BAUD               Baud Rate (default is 115200)
  --raw RAW             Binary file for raw message
  -f BINFILE            Binary file to program into the target device
  -i {0,1,2,3,4,5,6,7,32,255}
                        ImageType (0: SBL, 1: AM3P, 2: Patch, 3: Main, 4:
                        Child, 5: CustOTA, 6: NonSecure, 7: Info0 32:
                        Info0_NOOTA) 255: Invalid) - default[Invalid]
  -o OTADESC            OTA Descriptor Page address (hex) - (Default is
                        0xFE000 - at the end of main flash) - enter 0xFFFFFFFF
                        to instruct SBL to skip OTA
  -r {0,1,2}            Should it send reset command after image download? (0
                        = no reset, 1 = POI, 2 = POR) (default is 1)
  -a {0,1,-1}           Should it send abort command? (0 = abort, 1 = abort
                        and quit, -1 = no abort) (default is -1)
  --split SPLIT         Specify the max block size if the image will be
                        downloaded in pieces
```

7.1 Program INFO0 Using INFO0-NOOTA

Use the UART Wired Update script to (re)program INFO0 using the INFO0-NOOTA blob (generated as in *Section 5.1.2 Create INFO0-NOOTA Wired Update Image Blob on page 17*):

```
./uart_wired_update.py -b 115200 COM<X> -r 0 -f info0_wire.bin -i 32
```

7.2 Program Main Non-Secure Firmware

Use the UART Wired Update script to (re)program Main Firmware using the Non-Secure wire update blob (generated as in *Section 5.1.1 Create Non-Secure Wired Update Image Blob on page 17*)[1]:

```
./uart_wired_update.py -b 115200 COM<X> -r 1 -f main_nonsecure_wire.bin -i 6
```

7.3 Program Main Secure Firmware

Use the UART Wired Update script to (re)program Secure Main Firmware using the Secure wire update blob (generated as in *Section 5.1.1 Create Non-Secure Wired Update Image Blob on page 17*):

```
./uart_wired_update.py -b 115200 COM<X> -r 1 -f main_secure_swire.bin -i
3
```

7.4 Program SBL Upgrade Firmware

Use the UART Wired Update script to upgrade SBL Firmware using the SBL wire update blob (generated as in *Section 5.1.3 Create Secure Bootloader (SBL) Wired Update Image Blob on page 17*):

```
./uart_wired_update.py -b 115200 COM<X> -r 1 -f sbl_wire.bin -i 0
```

[1] The default command assumes last page of available flash to construct the OTA descriptor page, as required by the Upgrade process, as described in the*Apollo3 Blue Secure Update Flow User's Guide*. For non-default allocation of the OTA descriptor page, it can be specified using –o parameter.

7.5 Recover the Device

Use the UART Wired Update script to send Device Recover message (generated as in *Section 6.1.2 Create Non-Secure recover message on page 20*):

```
./uart_wired_update.py -b 115200 COM<X> -r 0 –o 0xFFFFFFFF --raw
recover.msg
```

7.6 Program Patch Upgrade

Use the UART Wired Update script to program patch using the patch wire update blob (generated as in *Section 5.1.5 Create Patch Wired Update Image Blob on page 18*):

```
./uart_wired_update.py -b 115200 COM<X> -r 1 -f patch_wire.bin -i 2
```

7.7 Upgrading Multiple Images in One Step

SBL supports upgrading multiple images in a single upgrade cycle using multiple entries in OTA Descriptor.

UART Wired Update scripts can be used to achieve the same. The script is to be run multiple times, once for each image. The key here is that OTA Descriptor is to be set only in the first invocation, and reset is to be issued only for the last one.

Example below shows upgrading an isolated data segments and main image (all considered non-secure main images generated as in *Section 5.1.1 Create Non-Secure Wired Update Image Blob on page 17*) together using **uart_wired_update.py**:

First image (also programs the OTA Descriptor, and does not reset the device):

```
./uart_wired_update.py -b 115200 COM<X> -f img1_nonsecure_wire.bin -i 6 -r 0
```

Second image (does not program the OTA Descriptor or reset the device):

```
./uart_wired_update.py -b 115200 COM<X> -f img2_nonsecure_wire.bin -i 6 -
r 0 –o 0xFFFFFFFF
```

Third image (does not program the OTA Descriptor but resets the device to initiate the upgrade):

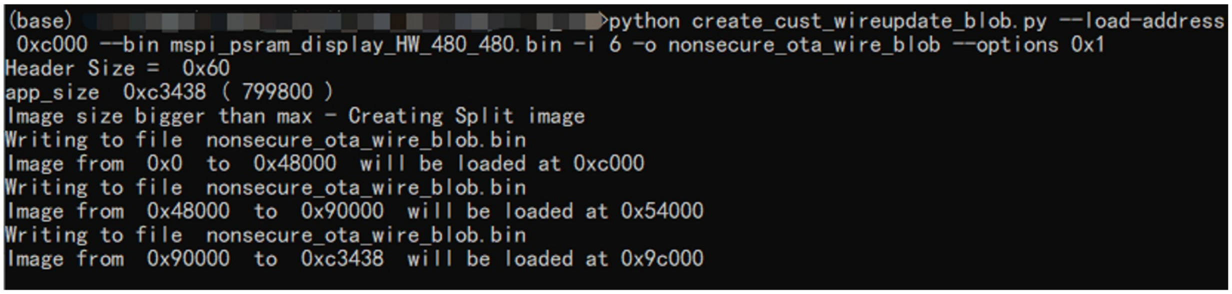
```
./uart_wired_update.py -b 115200 COM<X> -f img3_nonsecure_wire.bin -i 6 -
r 1 –o 0xFFFFFFFF
```

7.8 Upgrading Large Binary (Using --split feature)

The SBL reserves a ~96Kbytes SRAM area for its own operation and leaves a maximum usable SRAM for wired update the total SRAM size – 96Kbytes, (e.g., 288Kbytes in Apollo3 Blue). When the target user binary is larger than the usable SRAM size, the customer wired update blob binary will be automatically split into chunks of maximum usable SRAM size as default setting, (e.g., for Apollo3 Blue 288Kbytes per chunk).

For example, in the case below, a binary of 799800bytes is converted into a wired update blob, and is automatically split into 288Kbytes chunks.

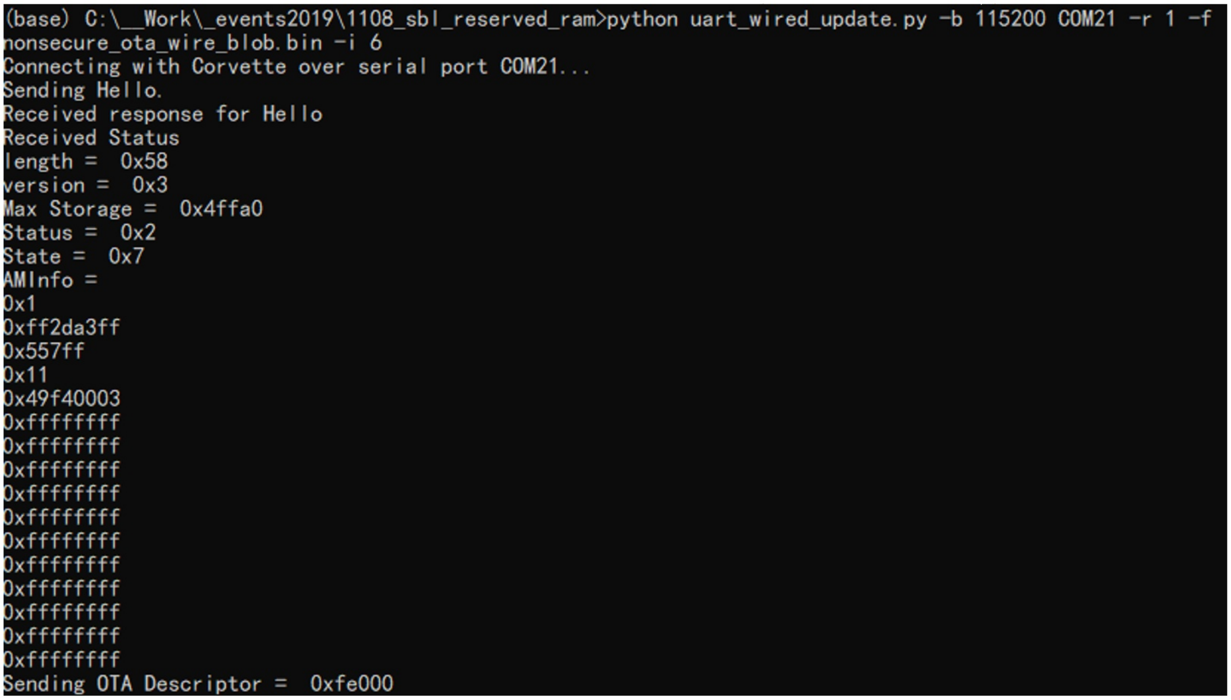
```
./create_cust_wireupdate_blob.py --load-address 0xc000 --bin <input_-
file_name> -i 6 -o <output_file_name> --options 0x1
```



```
(base) [redacted]>python create_cust_wireupdate_blob.py --load-address
 0xc000 --bin mspi_psram_display_HW_480_480.bin -i 6 -o nonsecure_ota_wire_blob --options 0x1
Header Size =  0x60
app_size  0xc3438 ( 799800 )
Image size bigger than max - Creating Split image
Writing to file  nonsecure_ota_wire_blob.bin
Image from  0x0  to  0x48000  will be loaded at 0xc000
Writing to file  nonsecure_ota_wire_blob.bin
Image from  0x48000  to  0x90000  will be loaded at 0x54000
Writing to file  nonsecure_ota_wire_blob.bin
Image from  0x90000  to  0xc3438  will be loaded at 0x9c000
```

User can later on use **uart_wired_update.py** load the target blob into the target device, e.g.,:

```
./uart_wired_update.py -b 115200 COM<X> -r 1 -f <target_file_name> -i 6
```



```
(base) C:\__Work\_events2019\1108_sbl_reserved_ram>python uart_wired_update.py -b 115200 COM21 -r 1 -f
nonsecure_ota_wire_blob.bin -i 6
Connecting with Corvette over serial port COM21...
Sending Hello.
Received response for Hello
Received Status
length =  0x58
version =  0x3
Max Storage =  0x4ffa0
Status =  0x2
State =  0x7
AMInfo =
0x1
0xff2da3ff
0x557ff
0x11
0x49f40003
0xffffffff
0xffffffff
0xffffffff
0xffffffff
0xffffffff
0xffffffff
0xffffffff
0xffffffff
0xffffffff
0xffffffff
0xffffffff
Sending OTA Descriptor =  0xfe000
```

NOTE: If the target device has Info0 setup with --sresv (SRAM reservation), user cannot leave the --split parameter as default. When generate target blob file and loading the blob file into the target device, --split parameter has to be specified. The maximum --split value is calculated as:

Total SRAM size - 96Kbytes - SRAM reservation size

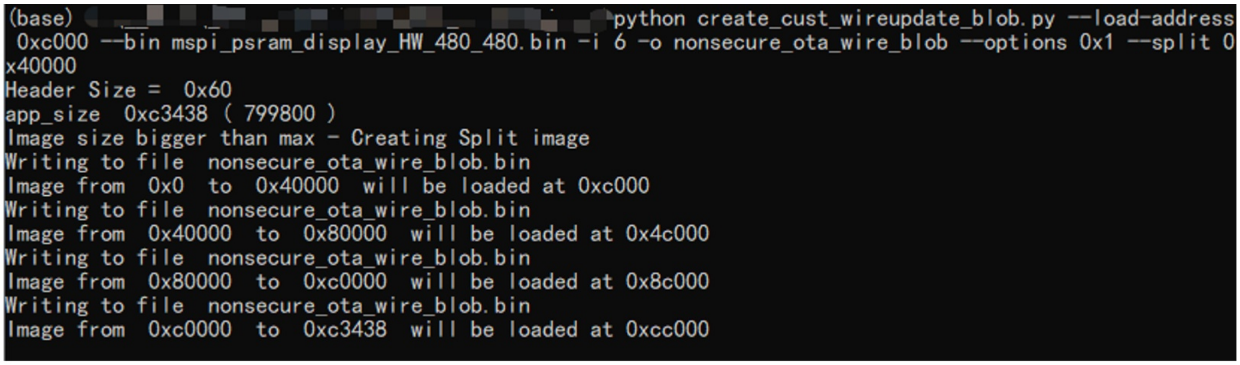
e.g., If the info0 specifies a 32Kbytes of SRAM reservation on Apollo3 Blue, the maximum --split value is:

384Kbytes - 96Kbytes - 32Kbytes = 256Kbytes (0x40000)

In such case, user should add --split 0x40000 parameter both for blob generation and for UART loading.

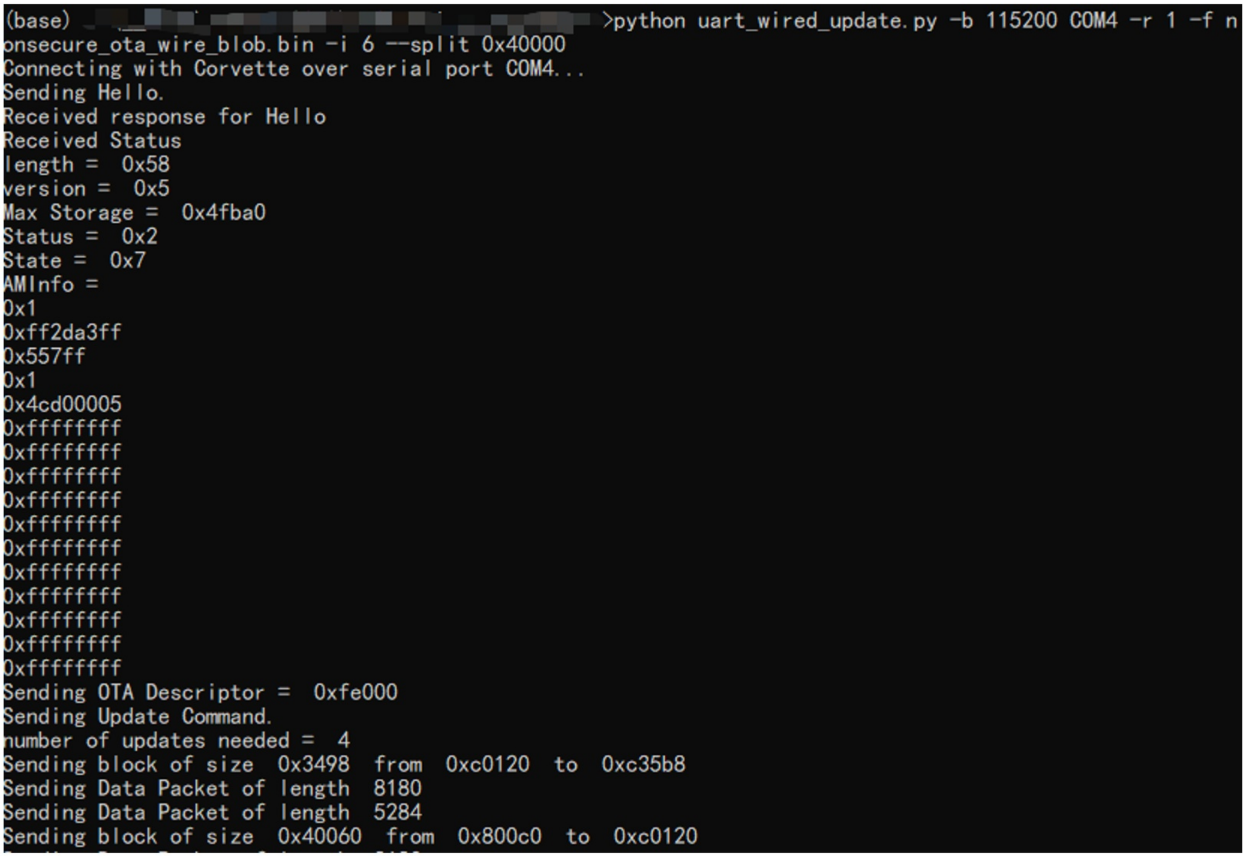
As shown below:

```
./create_cust_wireupdate_blob.py --load-address 0xc000 --bin <input_-
file_name> -i 6 -o <output_file_name> --options 0x1 --split 0x40000
```



```
(base) [redacted] python create_cust_wireupdate_blob.py --load-address
 0xc000 --bin mspi_psram_display_HW_480_480.bin -i 6 -o nonsecure_ota_wire_blob --options 0x1 --split 0
x40000
Header Size =  0x60
app_size  0xc3438 ( 799800 )
Image size bigger than max - Creating Split image
Writing to file  nonsecure_ota_wire_blob.bin
Image from  0x0  to  0x40000  will be loaded at 0xc000
Writing to file  nonsecure_ota_wire_blob.bin
Image from  0x40000  to  0x80000  will be loaded at 0x4c000
Writing to file  nonsecure_ota_wire_blob.bin
Image from  0x80000  to  0xc0000  will be loaded at 0x8c000
Writing to file  nonsecure_ota_wire_blob.bin
Image from  0xc0000  to  0xc3438  will be loaded at 0xcc000
```

```
./uart_wired_update.py -b 115200 COM4 -r 1 -f <target_file_name> -i 6 --
split 0x40000
```



```
(base) [redacted]>python uart_wired_update.py -b 115200 COM4 -r 1 -f n
onsecure_ota_wire_blob.bin -i 6 --split 0x40000
Connecting with Corvette over serial port COM4...
Sending Hello.
Received response for Hello
Received Status
length =  0x58
version =  0x5
Max Storage =  0x4fba0
Status =  0x2
State =  0x7
AMInfo =
0x1
0xff2da3ff
0x557ff
0x1
0x4cd00005
0xffffffff
0xffffffff
0xffffffff
0xffffffff
0xffffffff
0xffffffff
0xffffffff
0xffffffff
0xffffffff
0xffffffff
0xffffffff
Sending OTA Descriptor =  0xfe000
Sending Update Command.
number of updates needed =  4
Sending block of size  0x3498  from  0xc0120  to  0xc35b8
Sending Data Packet of length  8180
Sending Data Packet of length  5284
Sending block of size  0x40060  from  0x800c0  to  0xc0120
```

SECTION

8

OTA Update

Ambiq SDK provides an example OTA application AMOTA, which implements a specific transfer protocol with a counterpart host implemented as a Phone App (Ambiq_BLE App)

Script **ota_binary_converter.py** in **\tools\amota\scripts\** can be used to generate an OTA blob compatible to AMOTA. Most of the optional parameters are no longer relevant for Apollo3.

8.1 Example Usage

8.1.1 AMOTA update of NonSecure/Secure Main Firmware

Generate the OTA blob compatible to AMOTA using the Update Image (generated as in *Section 4.2.1.1 Create a Non-Secure Customer Image on page 15*):

```
./ota_binary_converter.py --appbin main_ota.bin -o main_ ota_amota
```

Thereafter, the normal procedure to upgrade the image using AMOTA and Ambiq_BLE App on the phone (AMOTA Example User's Guide) can be followed to upgrade the image on the device.

SECTION

9

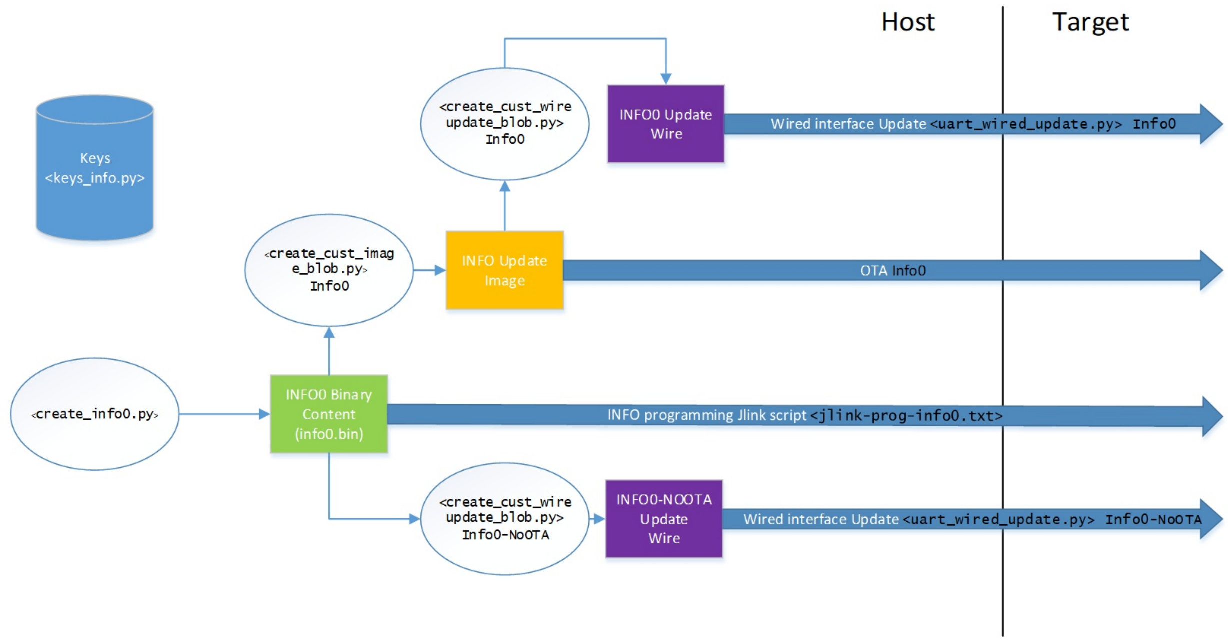
Programming Options and Usage of Scripts

This section depicts various options of programming the device, and how the scripts described in this document facilitate the same.

9.1 INFO0

- Script <create_info0.py> can be used to create an INFO0 binary.
- There are multiple ways thereafter to update Device Info0 to match with this generated binary.
  - Using Jlink Script <jlink-prog-info0.txt>
  - Using OTA
    - Generate OTA image using <create_cust_image_blob.py> with image type Info0
    - Update it over the air
- Using Wired Update
  - Two possible options:
    - No OTA - This will cause SBL to update Info0 bypassing the OTA processing
      - Create Wired Update Blob using <create_cust_wired_update_blob.py> with image type Info0-NoOTA
      - Use <uart_wired_update.py> to download using image type Info0-NoOTA
    - Process it through regular OTA processing (like other images)
      - Generate OTA image using <create_cust_image_blob.py> with image type Info0
      - Create Wired Update Blob using <create_cust_wired_update_blob.py> with image type Info0
      - Use <uart_wired_update.py> to download using image type Info0

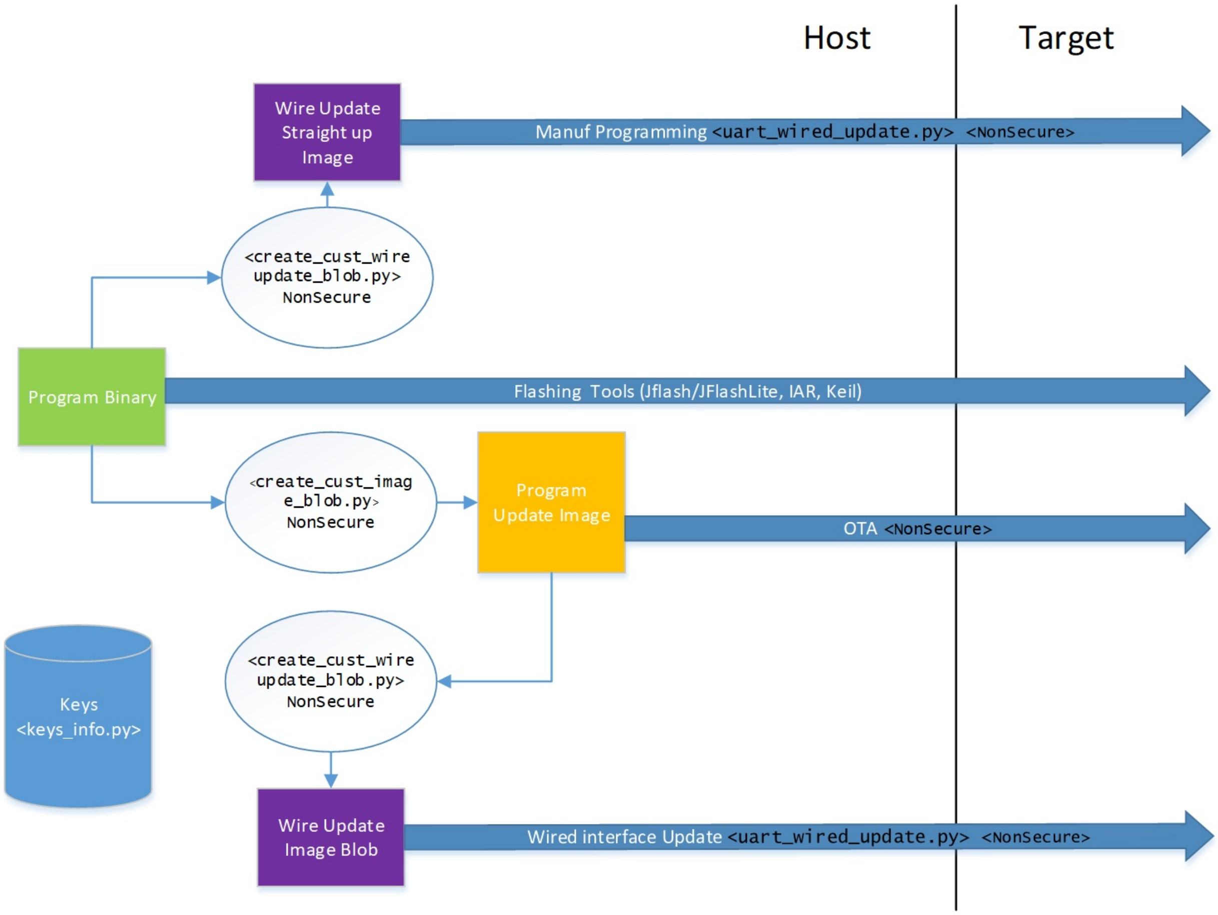
Figure 9-1: INFO0 Workflow


Host
Target
Keys <keys_info.py>
<create_cust_wire update_blob.py> Info0
INFO0 Update Wire
Wired interface Update <uart_wired_update.py> Info0
<create_cust_image_blob.py> Info0
INFO Update Image
OTA Info0
<create_info0.py>
INFO0 Binary Content (info0.bin)
INFO programming Jlink script <jlink-prog-info0.txt>
<create_cust_wire update_blob.py> Info0-NoOTA
INFO0-NOOTA Update Wire
Wired interface Update <uart_wired_update.py> Info0-NoOTA

9.2 Firmware Images or Data Binaries (Non-Secure)

- Generate Program Image using preferred toolchain (linked at 0xC000 or above [to match with INFO0 setting]).
- There are multiple ways thereafter to update the Device with this generated binary.
- Using Flashing Tools or IDEs to program the flash (use the generated image directly)
- Using OTA
  - Generate OTA image using <create_cust_image_blob.py> with image type NonSecure
  - Update it over the air
- Using Wired Update
  - Generate OTA image using <create_cust_image_blob.py> with image type NonSecure
  - Create Wired Update Blob using <create_cust_wired_update_blob.py> with image type NonSecure
  - Use <uart_wired_update.py> to download using image type NonSecure
- Manufacturing Programming
  - For the first time programming at manufacturing facility, the image could be directly loaded to the final install location without needing to go through the traditional OTA
  - Create Wired Update Blob using <create_cust_wired_update_blob.py> with image type NonSecure – set options as 0 (Disable OTA). Use load-address as the actual install address
  - Use <uart_wired_update.py> to download using image type NonSecure

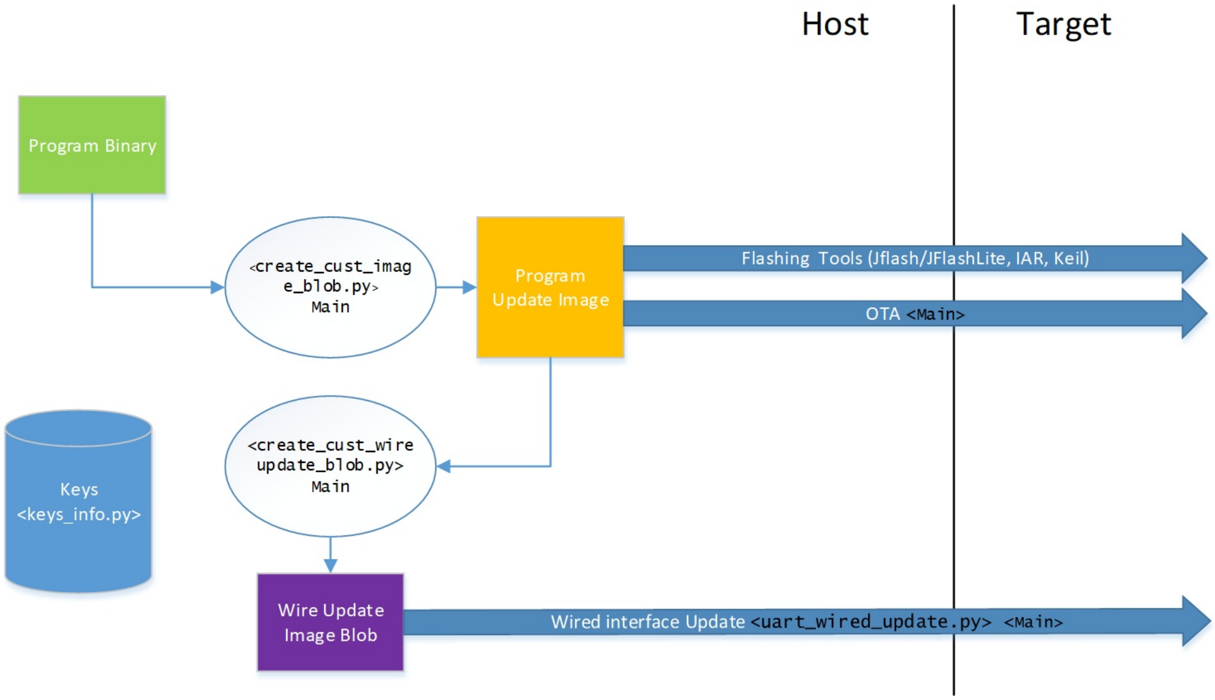
Figure 9-2: Firmware Images or Data Binaries (Non-Secure)


Host
Target
Wire Update Straight up Image
Manuf Programming <uart_wired_update.py> <NonSecure>
<create_cust_wire update_blob.py> NonSecure
Program Binary
Flashing Tools (Jflash/JFlashLite, IAR, Keil)
<create_cust_image_blob.py> NonSecure
Program Update Image
OTA <NonSecure>
Keys <keys_info.py>
<create_cust_wire update_blob.py> NonSecure
Wire Update Image Blob
Wired interface Update <uart_wired_update.py> <NonSecure>

9.3 Firmware Images or Data Binaries (Secure)

- Generate Program Image using preferred toolchain (linked at 0xC100 [match with INFO0 setting + 0x100]).
- Generate Update image using <create_cust_image_blob.py> with image type Main
  - This will create necessary headers needed by SBL for secure boot.
- There are multiple ways thereafter to update the Device with this generated update image.
- Using Flashing Tools or IDEs to program the flash (use the update image directly)
- Update it over the air (use the update image directly)
- Using Wired Update
  - Create Wired Update Blob using <create_cust_wired_update_blob.py> with image type Main
  - Use <uart_wired_update.py> to download using image type Main

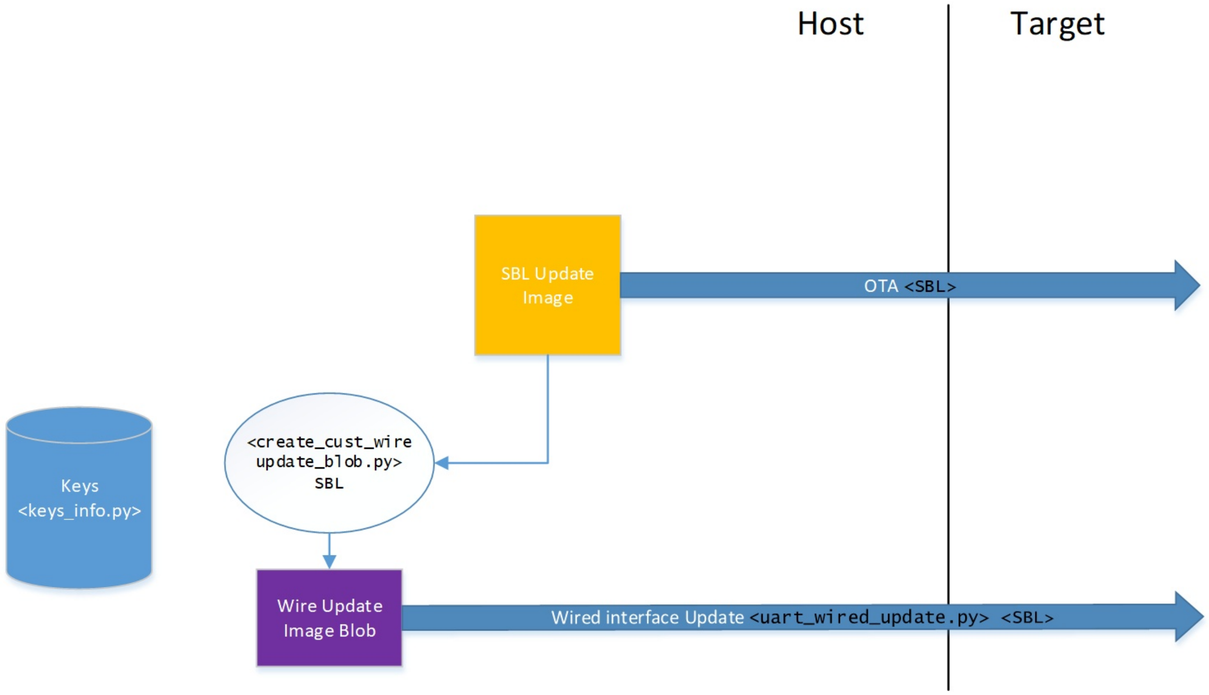
Figure 9-3: Firmware Images or Data Binaries (Secure)


Host
Target
Program Binary
<create_cust_image_blob.py> Main
Program Update Image
Flashing Tools (Jflash/JFlashLite, IAR, Keil)
OTA <Main>
Keys <keys_info.py>
<create_cust_wire update_blob.py> Main
Wire Update Image Blob
Wired interface Update <uart_wired_update.py> <Main>

9.4 SBL Update

- SBL update image is provided by Ambiq.
- There are multiple ways thereafter to update the Device with this generated update image.
  - Update it over the air (use the update image directly)
  - Using Wired Update
    - Create Wired Update Blob using <create_cust_wired_update_blob.py> with image type SBL
    - Use <uart_wired_update.py> to download using image type SBL

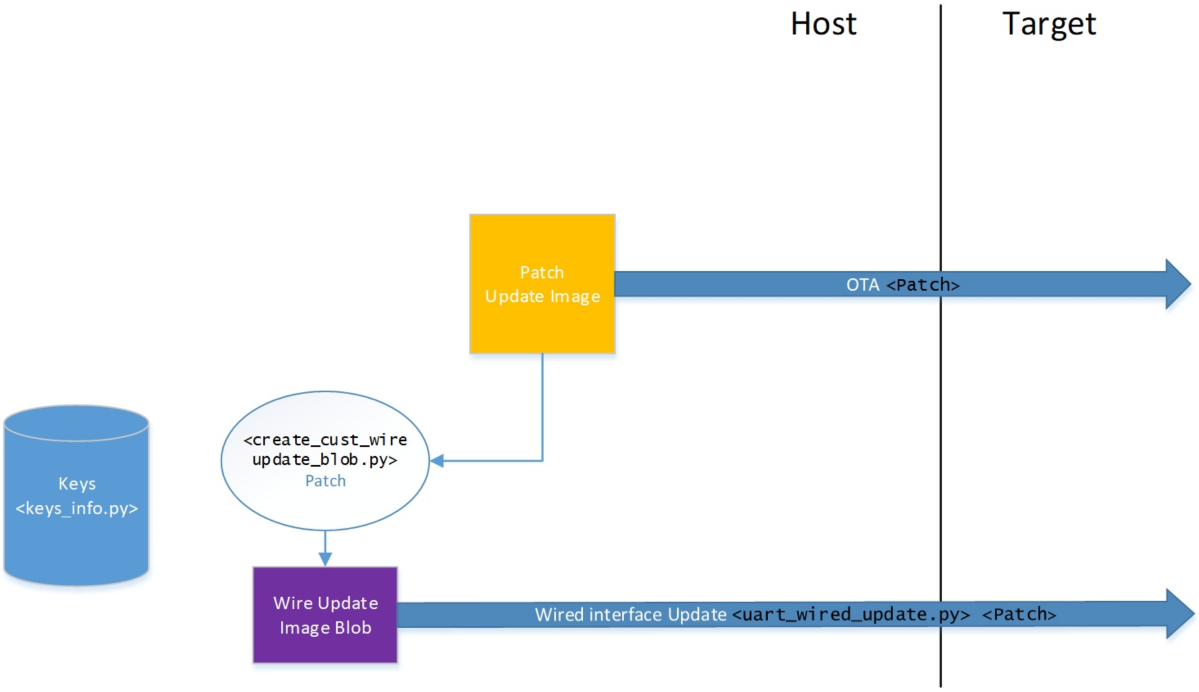
Figure 9-4: SBL Update


Host
Target
SBL Update Image
OTA <SBL>
Keys <keys_info.py>
<create_cust_wire update_blob.py> SBL
Wire Update Image Blob
Wired interface Update <uart_wired_update.py> <SBL>

9.5 Ambiq Patch Update

- Ambiq Patch update image is provided by Ambiq.
- There are multiple ways thereafter to update the Device with this generated update image.
  - Update it over the air (use the update image directly)
  - Using Wired Update
    - Create Wired Update Blob using <create_cust_wired_update_blob.py> with image type patch
    - Use <uart_wired_update.py> to download using image type patch

Figure 9-5: Ambiq Patch Update

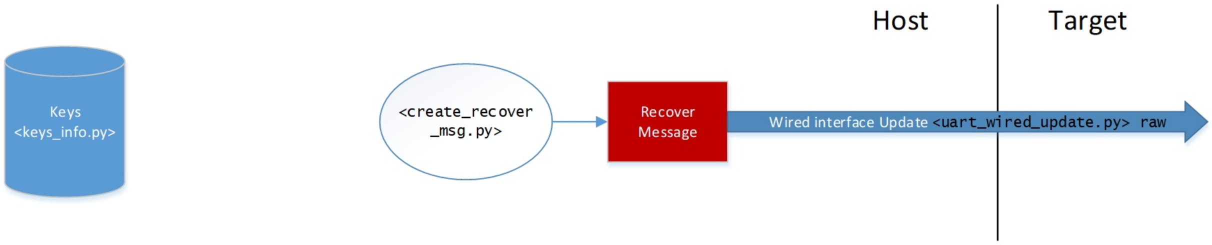


9.6 Device Recovery

9.6.1 Non-Secure Part

- Generate RECOVER message using <create_recover_msg.py>
- Use <uart_wired_update.py> to send the recover message using "raw" option

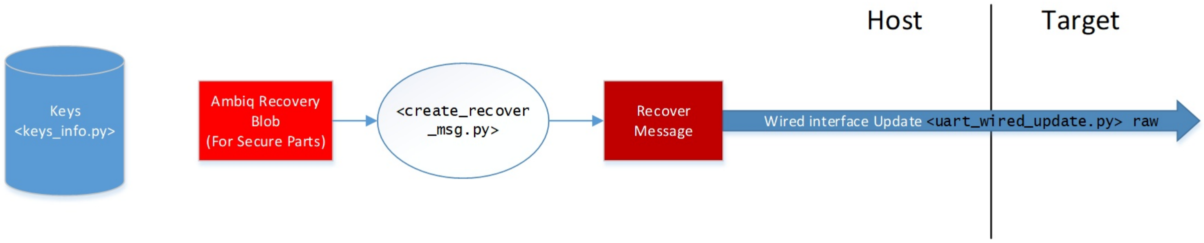
Figure 9-6: Non-Secure Part



9.6.2 Secure Part

- Contact Ambiq securely to get Ambiq Recovery Blob specific to the part(s).
- Generate RECOVER message using <create_recover_msg.py>, supplying the aforementioned blob image.
- Use <uart_wired_update.py> to send the recover message using "raw" option

Figure 9-7: Secure Part



SECTION

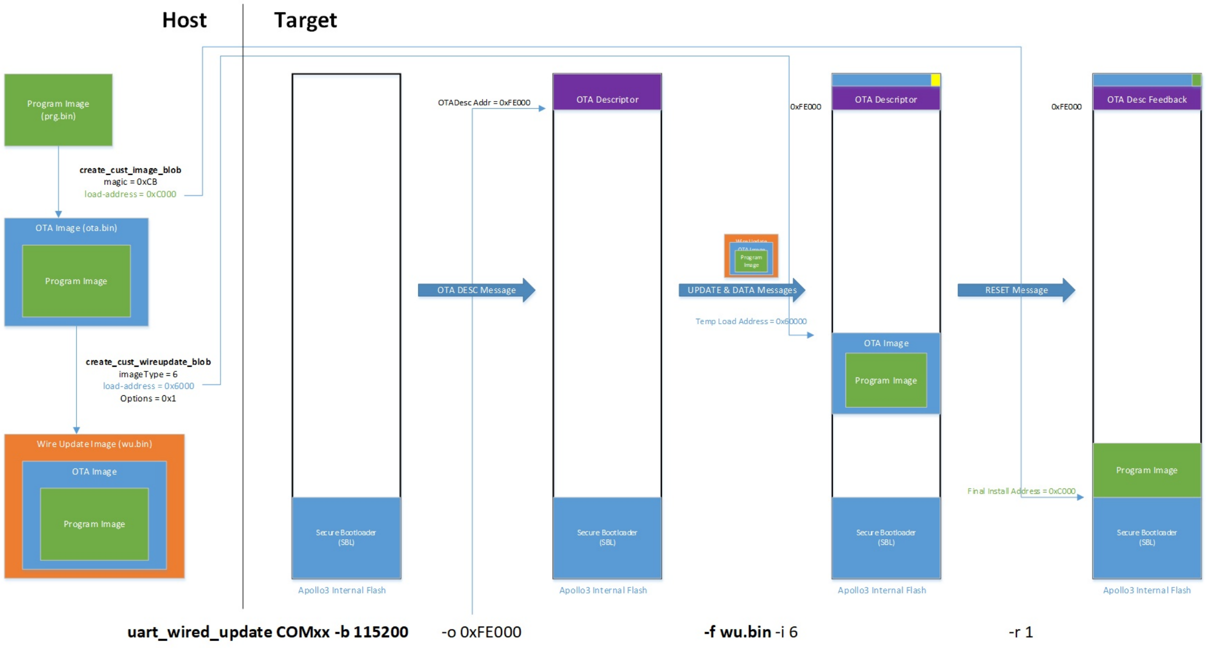
10 Example OTA Process Flow Using Scripts

Refer to *Apollo3 Blue Secure Update Flow User's Guide* for various image types, the corresponding header formats, and the overall design of the Secure OTA in Apollo3 Blue.

This section briefly describes how the sample scripts supplied with Ambiq Apollo3 SDK can be used to help with the OTA process.

- Generate the Program Image prg.bin (linked at 0xC000) using IDE of choice
- Create Corresponding OTA Image ota.bin using <create_cust_image_blob.py>
  - load-address (0xC000) specified indicates where the user wants the image to be installed in main flash at the end of OTA
- Create Wired Update Image Blob to prepare the OTA image for wired update, wu.bin using <create_cust_wireupdate_blob.py>
  - Set options to 0x1 to indicate to SBL to initiate an OTA for the downloaded image
  - load-address (0x60000) specified indicates where the user wants the image to be temporarily loaded in main flash before initiating OTA
    - For flash constrained systems, we allow this temp place to overlap with final install location, as long as the temp address is greater than or equal to install address) where you want to store the OTA image.
- Use script <uart_wired_update.py> to transfer the wired update blob, and instruct SBL to initiate OTA on the downloaded image
  - Option –o specifies where SBL can build the OTA Descriptor (Default is last page in flash)
    - Should point to a free page in flash
    - Cannot overlap with either the temporary load-address for the downloaded image, or the final install address for the main image
    - Cannot be located in a protected region of flash
  - This script configures the OTA Descriptor, downloads the Wired update blob and initiates the OTA of the same, following the process described in *Apollo3 Blue Secure Update Flow User's Guide*

Figure 10-1: Example of OTA Process Flow Using Scripts



```
uart_wired_update COMxx -b 115200 -o 0xFE000 -f wu.bin -i 6 -r 1
```

6500 River Place Boulevard, Building 7, Suite 200, Austin, TX 78730
www.ambiq.com
sales@ambiq.com
+1 (512) 879-2850

A-SOCA3B-UGGA01EN v3.1
October 2022