Apollo3-Blue
Secure Update Flow

Revision 1.3
Mar 2019

Revision History

Date	Revision	History	Reviser
Mar 22, 2018	0.1	Initial Version	J. Shah
Mar 29, 2018	1.0	Apollo3-Blue SDK Alpha Release	J. Shah
June 28, 2018	1.1	Further clarifications on OTA Descriptor & Device recovery Clarification on customer specific magic numbers	J. Shah
Feb 2, 2019	1.2	Updated to reflect the SBL-V3 functionality.	D. Munsinger
Mar 14, 2019	1.3	Updated Secondary Bootloader Section Updated Wired Update Sequence diagram	J. Shah

Table of Contents

1. Introduction

This document describes the methods, data formats and protocols supported by the Ambiq Secure Bootloader (SBL) for both In-Field and Over-the-Wire updates of software of the Apollo3-Blue MCU.

2. References

REF	Title	File
REF1	Apollo3-Blue Secure Update Flow	Apollo3-Blue_Secure_Update_Flow.pdf
REF2	AMOTA Example User's Guide	AMOTA_example_user's_guide.pdf
REF3	Apollo3-Blue Security Whitepaper	

3. General Security Features provided by Apollo3-Blue Secure Bootloader

3.1 Checksum

Each message/image contains a CRC32 checksum, used to ensure integrity. Standard 32b checksum function, as used in Ethernet FCS is used.

3.2 Authentication

To verify the authenticity of the originator of a message/image, a digital signature is computed and verified over the sensitive part of the body and verified against the known signature.

3.2.1 Two Signatures in the Image

The Standard Image formats for the OTA upgrades have provision for up to two HMAC signatures, one in cleartext, and one inside the encrypted blob. This allows flexibility in implementing either MAC-then-Encrypt, or Encrypt-then-MAC stategy. For most security, both the Signatures can be populated, which will allow for Dual verification during OTA install, and subsequent verification on each boot.

3.2.2 Extracting the Authentication Key

The message/image contains an Authentication key index. The key index is used to determine the Authentication Key from the applicable key bank (after any applicable unwrapping).

3.2.3 Authentication Algorithm

Currently only SHA256 HMAC signature is supported.
Signature computation is in accordance with standard HMAC using SHA256 as hash function:

```
SHA256-HMAC:
      ipad = the byte 0x36 repeated 32 times
      opad = the byte 0x5C repeated 32 times.

      Sign = H(K ⊕ opad, H(K ⊕ ipad, text))
```

Where H is the SHA256 hash function, and K is the 32 bytes (256 bits) secret key.

3.3 Decryption

To support confidentiality, each message/image provides for the sensitive information to be encrypted at origin. The image is then decrypted on the device as part of OTA processing.

3.3.1 Determining Encryption Key

The message/image contains an encrypted key (K_e), and a key index. The key index is used to determine a Key Encryption Key (KEK) from the KEK bank (after any applicable unwrapping).

The Key K used for decryption is determined using following operation:

$$K = F_{AES128\text{-}CBC}(K_e, KEK, IV)$$

IV is constructed as a sequence of 0's
Message/Image Decryption

3.3.2 Decryption Algorithm

We support Advanced Encryption Standard (AES) in the Cipher Block Chaining (CBC) mode.

Currently only Key size of 128b is supported.

$cleartext = F_{AESCBC}(ciphertext, K, IV)$

The IV is randomly selected for each message/image, and is communicated using the clear portion of the header along with the K_e.

3.4 Anti-Rollback

Each image header contains a version field. A configurable security policy allows mandatory enforcement of anti-rollback, where any OTA upgrade of an image with a version same or lower than that of the current image is rejected.

3.5 Key Unwrapping

To limit the potential damage if a particular device were to be physically attacked, exposing the stored keys, the underlying design supports a key wrapping mechanism using which the programmed values can be bound to a part, and hence not usable across other devices.

The stored key (K_w) is bound to the part by mean of unique 64b ChipID.

The programmed values are determined by wrapping the real keys at the time of manufacturing, and are unwrapped on the device as part of security operations to extract the real key.

$K = F_{unwrap}(K_w, K_{unwrap}, ChipID)$

Unwrapping function F_{unwrap} is configurable, and K_{unwrap} is the key used for unwrapping.

Fort best wrapping security, AES128-CBC can be used as a Key wrapping function.

$K = F_{AES128-CBC}(K_w, K_{unwrap}, IV)$

- IV represents Initialization Vector – which is generated from the 64 bit ChipID
    - $IV = (ChipID \mid ChipID << 64)$
- Here K_{unwrap} is the unique unwrapping key
    - For INFO0 customer key storage area, this the customer key (CK) programmed by the customer

3.6 Key Revocation

The key storage area is organized as multiple 128b slots, each identified using a key index.
Key Index 0-7 are reserved for Keys in Ambiq key storage area, and key index 8-15 represent the keys in the INFO0.

Corresponding to each key slot in the INFO area, there is a revocation mask using which the programmed keys can be revoked in field if a breach is discovered.

Each bit in the revocation mask (starting with msb) correspond to a key index, and if cleared to 0 – indicates that the key has been revoked and hence no longer acceptable for security validation.

Separation revocation masks are provisioned for Authentication Keys and the Key Encryption Keys (KEK) used for decryption.

3.7 Image Protection

On successful installation through SBL, the image header can also be used to direct the SBL to apply image protection features to the corresponding flash blocks. The flash blocks are specified in 16K granularity.

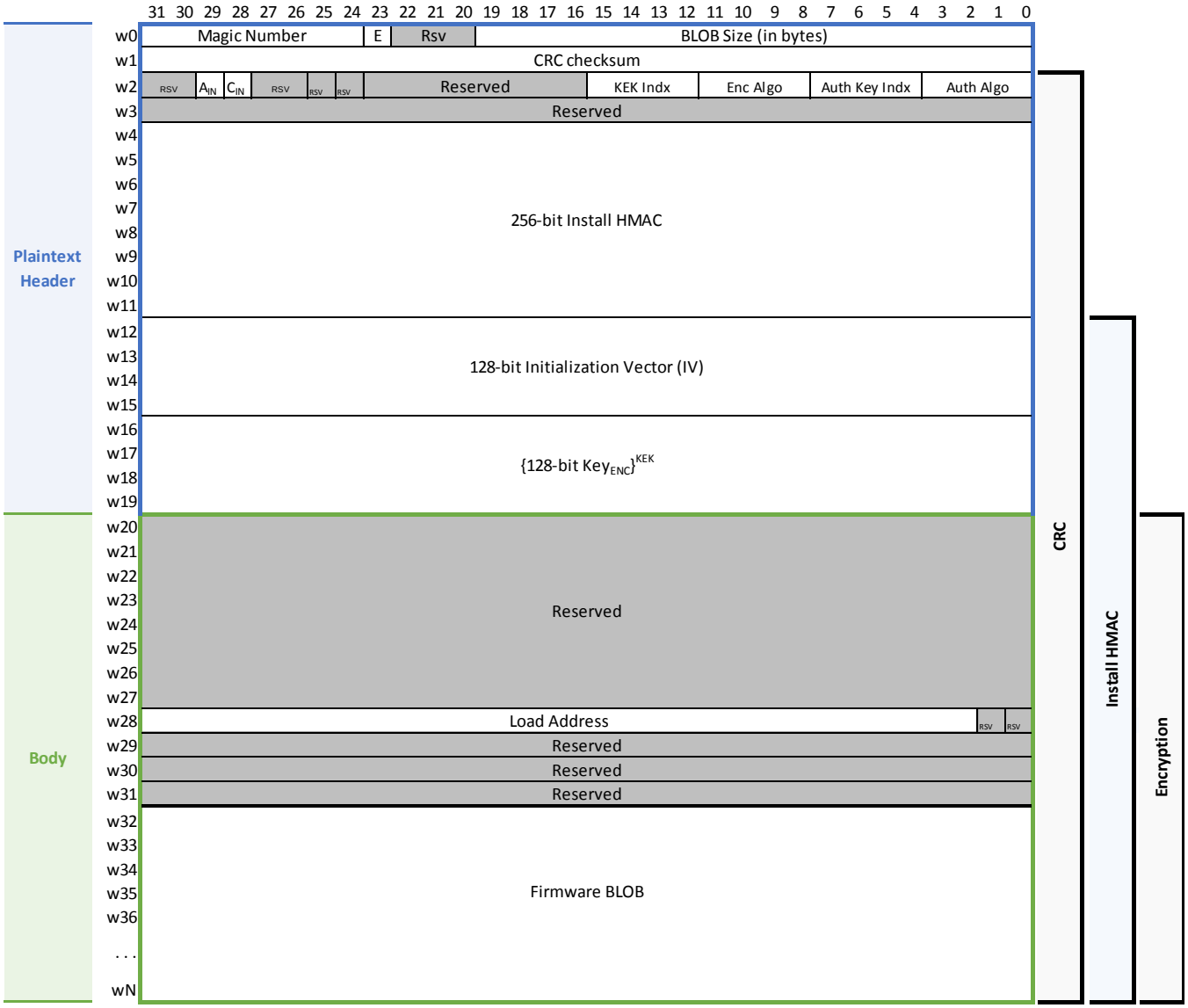
The blocks can be marked as “Write-Protected” to prevent overwriting the images either intentionally using malicious programs, or unintentionally. Such Write-protected images can be upgraded only through SBL by maintaining the secure upgrade trust chain.

The blocks can be marked as “Copy-Protected” to prevent Read access to the pages. This can be used to avoid exposing sensitive algorithms or programs from prying eyes. Care must be taken when using this feature to protect executable code – so as to generate the code using appropriate tools options so that there are no data reads in the code memory.

4. Apollo3 Image Headers

4.1 NonSecure Main/Data

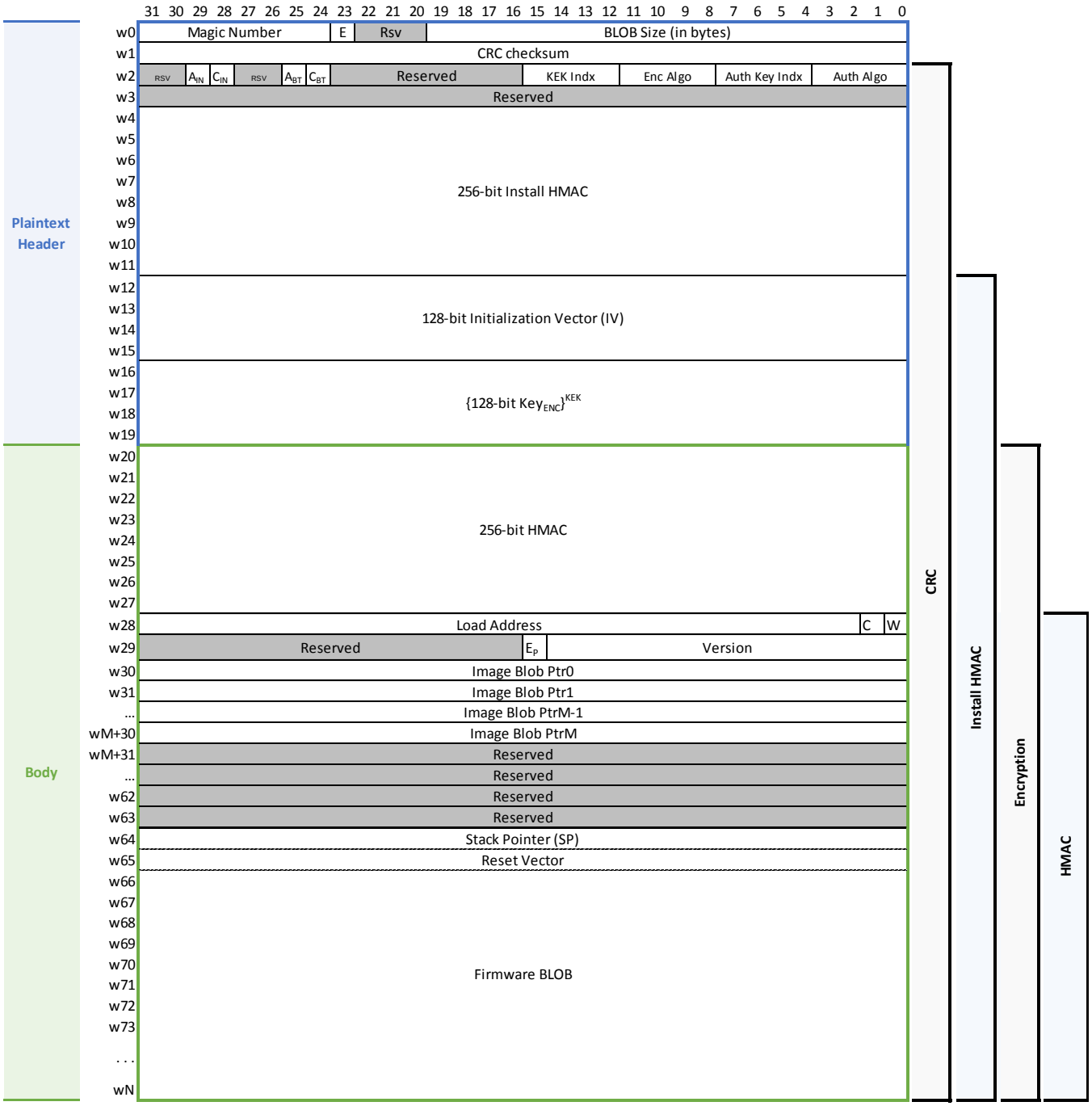
This is the image format for the non-secure image. Note that as part of install, the header is discarded.



Magic Number	Unique 8-bit value used to reference a valid image type. Non-secure firmware image type magic number is 0xCB.
E	Encrypt bit - When '1', the image must be decrypted
BLOB Size	Size of the image blob body (in bytes)
CRC Checksum	CRC checksum signature for the image. CRC is computed over the "CRC" region marked on the decrypted blob.
A_{IN}	Install-Authenticate enable bit - When '1', the image must be authenticated on installation
C_{IN}	Install-CRC enable bit - When '1', the image must be CRC verified on installation
KEK Indx	Key-Encryption-Key Index - this index specifies which KEK should be used within the KEK bank for all key unwrap functions
Enc Algo	Encryption Algorithm - specifies which algorithm is to be used for decrypting the image 0: N/A 1: AES-128 CBC others: not supported
Auth Key Indx	Authentication Key Index - this index specifies which authentication key should be used within the Auth Key bank for authentication of this image
Auth Algo	Authentication Algorithm - specifies which algorithm is to be used for authentication the image 0: N/A 1: SHA-256 others: not supported
256-bit Install HMAC	256 bit HMAC signature of the Install Blob following
128-bit IV	128-bit Initialization Vector used for seeding the encryption/decryption of the image
$\{128\text{-bit } Key_{ENC}\}^{KEK}$	128-bit KEK wrapped encryption key
256-bit HMAC	256 bit HMAC signature of the clear blob following
Load Address	Specifies the address where the image is to be installed (applicable to OTA only)
Version	Version number for the image

4.2 Secure Main

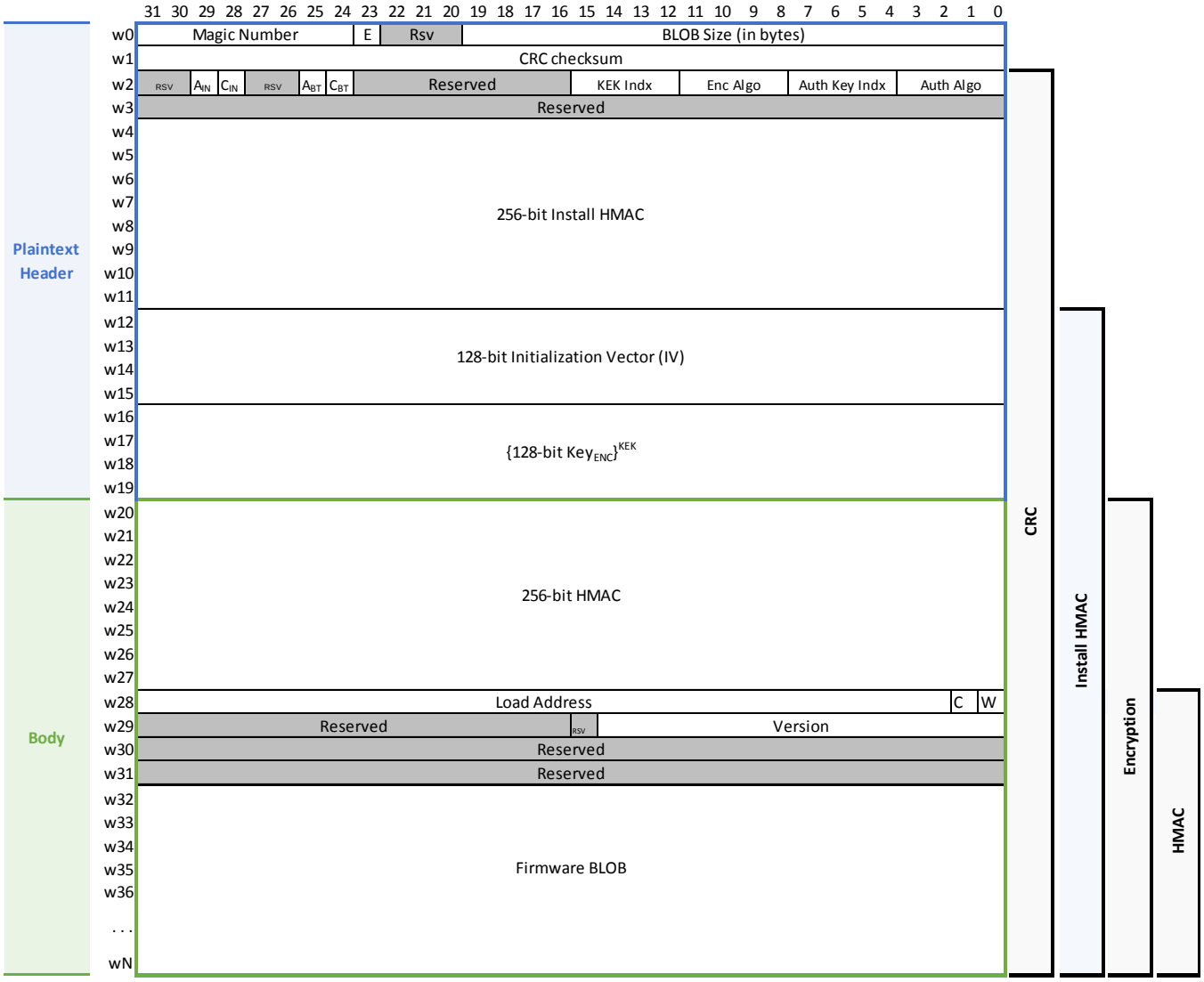
This is the image format for the Secure main firmware image. The headers are left intact in the flash, for SBL to be able to verify the image on each bootup time.



Magic Number	Unique 8-bit value used to reference a valid image type. Customer sbl_main or main firmware image type magic number is 0xC0.
BLOB Size	Size of the image blob body (in bytes)
CRC Checksum	CRC checksum signature for the image. CRC is computed over the "CRC" region marked on the decrypted blob.
E	Encrypt bit - When '1', the image must be decrypted
A_{IN}	Install-Authenticate enable bit - When '1', the image must be authenticated on installation
C_{IN}	Install-CRC enable bit - When '1', the image must be CRC verified on installation
A_{BT}	Boot-Authenticate enable bit - When '1', the image must be authenticated at boot time
C_{BT}	Boot-CRC enable bit - When '1', the image must be CRC verified at boot time
KEK Indx	Key-Encryption-Key Index - this index specifies which KEK should be used within the KEK bank for all key unwrap functions
Enc Algo	Encryption Algorithm - specifies which algorithm is to be used for decrypting the image 0: N/A 1: AES-128 CBC others: not supported
Auth Key Indx	Authentication Key Index - this index specifies which authentication key should be used within the Auth Key bank for authentication of this image
Auth Algo	Authentication Algorithm - specifies which algorithm is to be used for authentication the image 0: N/A 1: SHA-256 others: not supported
256-bit Install HMAC	256 bit HMAC signature of the Install Blob following
128-bit IV	128-bit Initialization Vector used for seeding the encryption/decryption of the image
$\{128\text{-bit Key}_{ENC}\}^{KEK}$	128-bit KEK wrapped encryption key
256-bit HMAC	256 bit HMAC signature of the clear blob following
Load Address	Specifies the address where the image is to be installed (applicable to OTA only)
C	Copy Protect - When '1', this bit indicates that the image should be copy protected after installation
W	Write Protect - When '1', this bit indicates that the image should be write protected after installation
E_P	Erase-Previous - When '1', this bit indicates that the previous image should be erased as part of OTA
Image Blob Ptr	Pointer(s) to a secondary image blob
Version	Version number for the image

4.3 Secure Child/Data

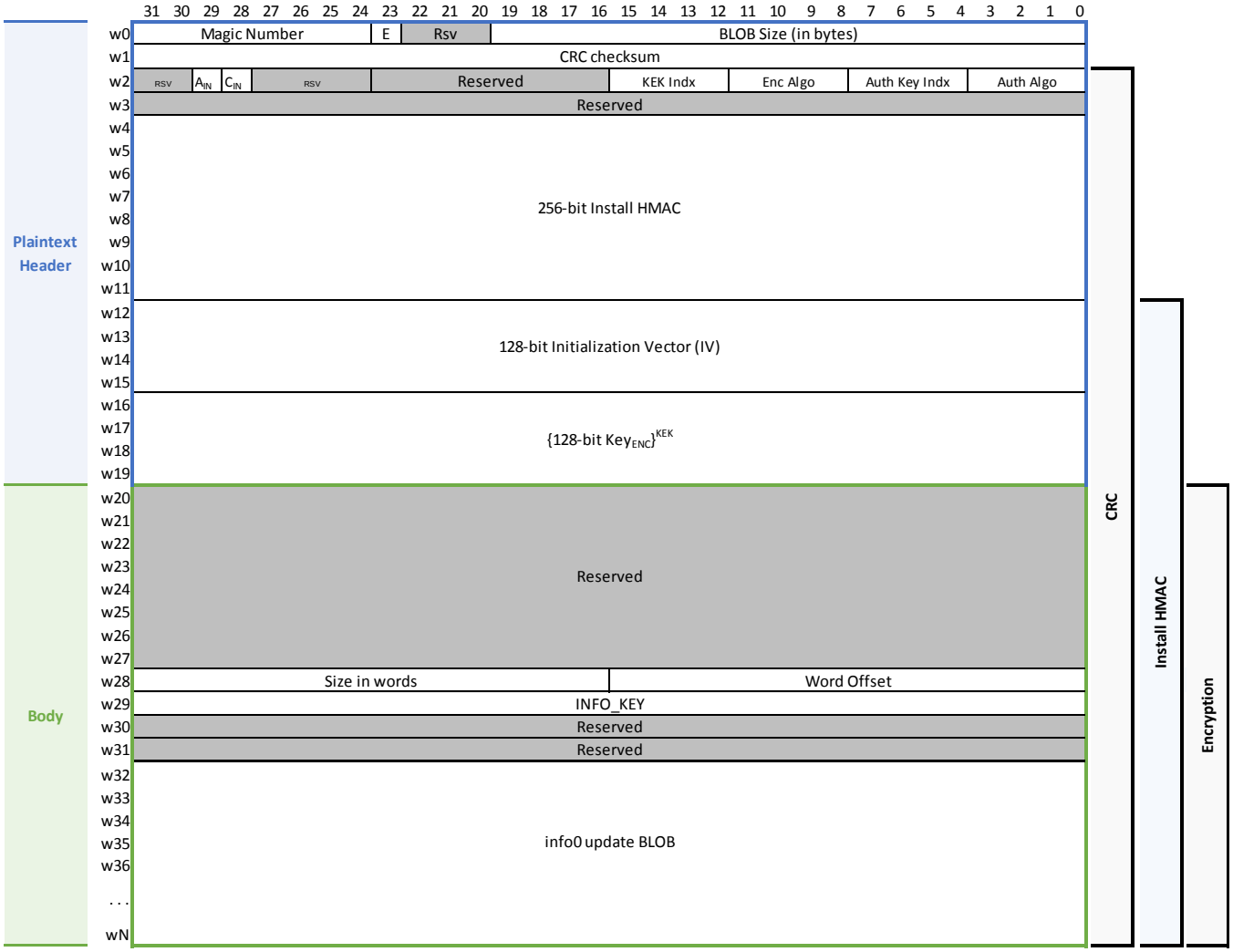
This is the image format for the Secure child firmware or data image. The headers are left intact in the flash, for SBL to be able to verify the image on each bootup time (if referenced by main).



Magic Number	Unique 8-bit value used to reference a valid image type. Customer child firmware image type magic number is 0xCC.
E	Encrypt bit - When '1', the image must be decrypted
BLOB Size	Size of the image blob body (in bytes)
CRC Checksum	CRC checksum signature for the image. CRC is computed over the "CRC" region marked on the decrypted blob.
A_{IN}	Install-Authenticate enable bit - When '1', the image must be authenticated on installation
C_{IN}	Install-CRC enable bit - When '1', the image must be CRC verified on installation
A_{BT}	Boot-Authenticate enable bit - When '1', the image must be authenticated at boot time
C_{BT}	Boot-CRC enable bit - When '1', the image must be CRC verified at boot time
KEK Indx	Key-Encryption-Key Index - this index specifies which KEK should be used within the KEK bank for all key unwrap functions
Enc Algo	Encryption Algorithm - specifies which algorithm is to be used for decrypting the image 0: N/A 1: AES-128 CBC others: not supported
Auth Key Indx	Authentication Key Index - this index specifies which authentication key should be used within the Auth Key bank for authentication of this image
Auth Algo	Authentication Algorithm - specifies which algorithm is to be used for authentication the image 0: N/A 1: SHA-256 others: not supported
256-bit Install HMAC	256 bit HMAC signature of the Install Blob following
128-bit IV	128-bit Initialization Vector used for seeding the encryption/decryption of the image
$\{128\text{-bit Key}_{ENC}\}^{KEK}$	128-bit KEK wrapped encryption key
256-bit HMAC	256 bit HMAC signature of the clear blob following
Load Address	Specifies the address where the image is to be installed (applicable to OTA only)
C	Copy Protect - When '1', this bit indicates that the image should be copy protected after installation
W	Write Protect - When '1', this bit indicates that the image should be write protected after installation
Version	Version number for the image

4.4 Info0-OTA

This is the image format for the OTA message for INFO0 update.



Magic Number	Unique 8-bit value used to reference a valid image type. Info0 update image type magic number is 0xCF.
E	Encrypt bit - When '1', the image must be decrypted
BLOB Size	Size of the image blob body (in bytes)
CRC Checksum	CRC checksum signature for the image. CRC is computed over the "CRC" region marked on the decrypted blob.
A_{IN}	Install-Authenticate enable bit - When '1', the image must be authenticated on installation
C_{IN}	Install-CRC enable bit - When '1', the image must be CRC verified on installation
KEK Indx	Key-Encryption-Key Index - this index specifies which KEK should be used within the KEK bank for all key unwrap functions
Enc Algo	Encryption Algorithm - specifies which algorithm is to be used for decrypting the image 0: N/A 1: AES-128 CBC others: not supported
Auth Key Indx	Authentication Key Index - this index specifies which authentication key should be used within the Auth Key bank for authentication of this image
Auth Algo	Authentication Algorithm - specifies which algorithm is to be used for authentication the image 0: N/A 1: SHA-256 others: not supported
256-bit Install HMAC	256 bit HMAC signature of the Install Blob following
128-bit IV	128-bit Initialization Vector used for seeding the encryption/decryption of the image
{128-bit Key_{ENC}}^{KEK}	128-bit KEK wrapped encryption key
Word Offset	Specifies the offset (in 4 byte multiple) in the infospace to be updated
Size in words	Specifies the size (in 4 byte multiple) of the infospace update blob
INFO_KEY	32b key required for Infospace programming

5. Apollo3 Secure In-Field Image Upgrade Flow

Apollo3 supports a secure in-field image upgrade flow using the pre-flashed Secure Bootloader (SBL). Ambiq Secure Bootloader has access to the key storage in the InfoSpace and can be used to optionally decrypt, authenticate and validate upgrade images before installing them. If configured so, only properly signed and protected images would be allowed in an update.

Ambiq Secure Bootloader can be used to securely update Secure Bootloader itself, Ambiq provided pre-installed libraries, and the main customer image (could be a secondary bootloader). Ambiq bootloader can also be leveraged to upgrade third party libraries in the flash.

In addition, the SBL supports non-secure updates when so configured in InfoSpace.

The following sections lists out steps in the secure in-field image update flow.

5.1 Image Download

The In-Field upgrade process is initiated by a user application downloading an image blob to the flash. This part of the upgrade process is specific to individual deployment scenarios and the user application implementation and is left to the customers. The Secure Upgrade framework does not mandate any specifics for this process. Depending on the deployment model, the image download could happen over traditional wired interfaces e.g. SPI/I2C or wirelessly OTA ("Over the Air") using BLE. The Upgrade application running on Apollo3 and its counterpart on the host/cloud side could implement their own protocol to ensure integrity, secrecy and authenticity of the image blob itself.[1]

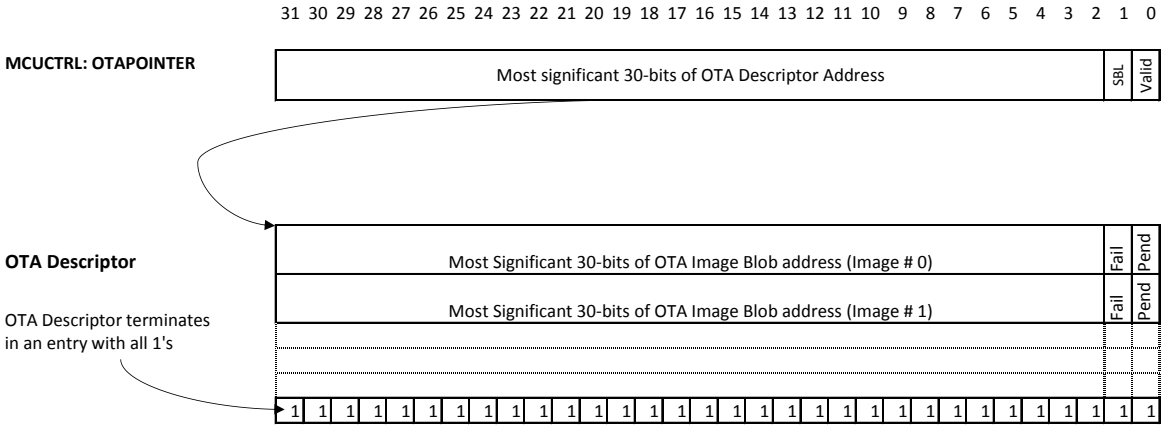
5.2 OTA Descriptor & OTA Pointer

The download application builds an OTA Descriptor containing the information about the upgrade image blob(s) corresponding to the update requests.

OTA Descriptor needs to be built at a page aligned address in flash.

The framework supports more than one image to be upgraded in same operation. The OTA Descriptor consists of list of pointers to the image blobs, with a special End of List Marker. This allows the upgrade application to update more than one images in one boot cycle, e.g. the upgrade application could construct a single OTA Descriptor to upgrade main firmware as well as certain third party library images located separately in the flash.

The download application communicates the OTA Descriptor information with the Secure Bootloader by means of a special register, REG_MCU_CTRL_OTAPOINTER.



OTAPOINTER will be initialized as follows:

[1] Ambiq Secure Bootloader by itself does not provide means to download the images from a host over BLE. This functionality, if required, must be built as part of a separate user space application.

- Most significant 30 bits will correspond to most significant 30 bits of OTA Descriptor
- Least Significant bit (bit 0) should be initialized to 1 to indicate a valid OTA Descriptor
- bit 1 should be initialized to 1 if the list contains an SBL OTA

OTA Descriptor points to a list of entries, each corresponding to an OTA blob, list terminating in 0xFFFFFFFF. Each list entry word comprises of following:

- Most significant 30 bits will correspond to most significant 30 bits of OTA blob pointer
- Blob pointer needs to be aligned to Flash Page boundary (8K)
- Least Significant 2 bits should be initialized to '11' to indicate a valid OTA Pending

After the Secure Bootloader processes an OTA, it clears the least significant bit (bit 0)

- Bit 1 indicates the status of the OTA: 0 for Success, 1 for Failure
- If Bit 0 is still set to 1 (Pending) after the OTA, this implies some other error (e.g. invalid or improperly formed OTA descriptor list) caused SBL to not process this OTA.
    - The OTA can be retried after clearing the issue

5.3 Reset

After accepting the required updates, the SBL or the download application constructs the OTA Descriptor and initializes the REG_MCU_CTRL_OTAPOINTER register accordingly. The actual Update is only initiated on the next Reset, which kicks in the bootloader.

5.4 Upgrade verification

As part of boot process, the secure bootloader inspects REG_MCU_CTRL_OTAPOINTER for any updates to be processed. If present, each update blob is processed as per the configured security policy.

- The security policy can be configured (via InfoSpace) to mandate Authentication to ensure only properly signed images would be accepted.
- The Secure Bootloader also supports encrypted image blobs, and the same can also be mandated by the security policy.

Ambiq Secure Bootloader enforces the configured security policy and validates the image blobs against the security assets in InfoSpace.

After optional decryption and authentication, if the image is found to be good, the bootloader then proceeds with installation of the image.

Secure Bootloader can also check for validity of third party libraries before installing the main image, if such dependencies are called for in the main image header.

5.5 Installation

A validated OTA image is installed to its designated place by the Secure Bootloader. Optionally, the Secure Bootloader can also be instructed to apply protection attributes (Copy and/or Write Protection) to the installed image.

5.6 Feedback

The OTA flow also allows for a feedback to the user application using the same OTA Descriptor – to communicate the OTA status of individual images back to the initiating application. This is accomplished using the same OTA Descriptor, as described in section 5.2.

5.7 Advanced Upgrade features

Apollo3 Secure Upgrade framework also includes provisions for extensions beyond the basic Upgrade as described above. Following is a list of some of these possibilities.

5.7.1 Integrating Customer Specific Bootloader into the Upgrade flow

Infospace settings provide provisions (SECURITY.PLONEXIT) to allow for the customer key area to be kept open for secondary bootloader to access for any customer specific image validation, or to apply any further image protections in the flash. This allows customers to implement advanced features (e.g. external flash support, different encryption or authentication schemes) which are not included in SBL using their own customer bootloader.

In addition, Ambiq Bootloader keeps the “unknown” image blobs (identified by magic number in image header) intact in the OTA descriptor, and passes them on to the Customer Bootloader. This allows for Customer Bootloader to leverage on the same Upgrade flow, when using advanced Authentication/Encryption Policies not supported in Ambiq Secure Bootloader. Currently following magic numbers can be used by the Customers for defining their own images: 0xC1 to 0xCA, 0xCD & 0xCE

It is the responsibility of the secondary bootloader to lock the infospace and the flash protection register access before passing control to the main firmware, if so desired.

5.7.2 Means to implement “Active/Standby”

For the main firmware image (or the customer bootloader image, if present), an optional scheme can be implemented which allows for maintaining dual images on the device – marked as Active or Standby.

Ambiq Bootloader relies on infospace fields (MAIN_PTR0 & MAIN_PTR1, along with MAIN_CNT_INDXCTR) to determine the location of the main firmware image.

5.7.3 Means to implement “Try before Install”

For the main firmware image (or the customer bootloader image, if present), an optional scheme can be implemented which allows for temporary install of the new image – which would then be made permanent only if it runs satisfactorily. A handshake mechanism between the new image and the Secure Bootloader allows for the new image to self-validate itself and instruct the Secure Bootloader to make the install permanent during next bootup.

Ambiq Bootloader relies on infospace fields (MAIN_PTR0 & MAIN_PTR1, along with MAIN_CNT_INDXCTR) to determine the location of the main firmware image. When doing an image upgrade of the main firmware, it is possible to install it at the alternate location. SBL will boot to the newly installed image, which can then do appropriate sanity checks before updating the MAN_CNT_INDXCTR to instruct SBL to boot to this alternate image for subsequent boots.

6. Apollo3 Wired update/recovery flow

Apollo3 SBL provides support for an external host to connect during bootup to upgrade images or to recover a failing device. SBL initiates Recovery if any of the boot time validations fail or if there is no valid image to boot to. Override is a feature provided by SBL, using which a forced image upgrade can be initiated using specific GPIO settings during the boot up.

The SBL uses the INFO0 configuration to determine the interfaces to examine (see SECURITYWIREDCFG : IFC). It first looks for UART connection, then the IOS connection of either SPI then I2C (these are mutually exclusive). The UART connection is associated with the timeout (see SECURITYWIREDCFG : TIMEOUT) while the SBL waits for the initial HELLO packet from the Host. The IOS interface is polled and the SBL uses the Slave Interrupt pin (see SECURITYWIREDCFG : SLVINTPIN) to indicate to the host that it is ready to receive packets. In this case it only waits for a fixed timeout of 500msec before exiting the update process.

In all cases, an external host needs to follow a predefined messaging protocol to instruct SBL to upgrade assets on the device. The following figures illustrates a high level message exchange to initiate an “upgrade” using prevalent security policies of the device.

Figure 1 shows the process for update/recovery when using the UART. In this case the DATA packets are sent as up to 8KB packets. The process starts with a HELLO/STATUS exchange within the timeout period. It ends with the RESET/ACK exchange which sends the SBL into SWPOR or SWPOI reset.

Figure 2 shows the process for update/recovery when using the IOS as SPI or I2C interface. The process starts with the Host resetting the Apollo3-Blue using nRST signal. Once initialization of the configured interface is complete, the SBL raises the Slave Interrupt pin to indicate it is ready to accept packets over the IOS Direct memory interface. All packets from the Host to the SBL are limited to 120 bytes due to the maximum LRAM memory size. This means the packets from the Host must be disassembled and then reassembled by the SBL-V3. Disassembly is done by adding a 32-bit header to each packet as follows:

```
typedef struct
{
uint32_t                    length  : 16;
uint32_t                    resv    : 14;
uint32_t                    bEnd    : 1;
uint32_t                    bStart  : 1;
} am_secboot_ios_pkthdr_t;
```

The start and end flags indicate the packets that represent the first and last of an original SBL wired protocol packet. The original packet data follows up to 116 bytes. The 116-byte Data Fragments compose an entire original DATA message and are stored by SBL-V3 up to 8KB. There is limited error checking except that the total bytes received must equal the length in the original message or it will be rejected.

Note that because the FIFO mode is used for Slave to Host transfers, the message size limit is 1023 bytes so disassembly/reassembly is not required.

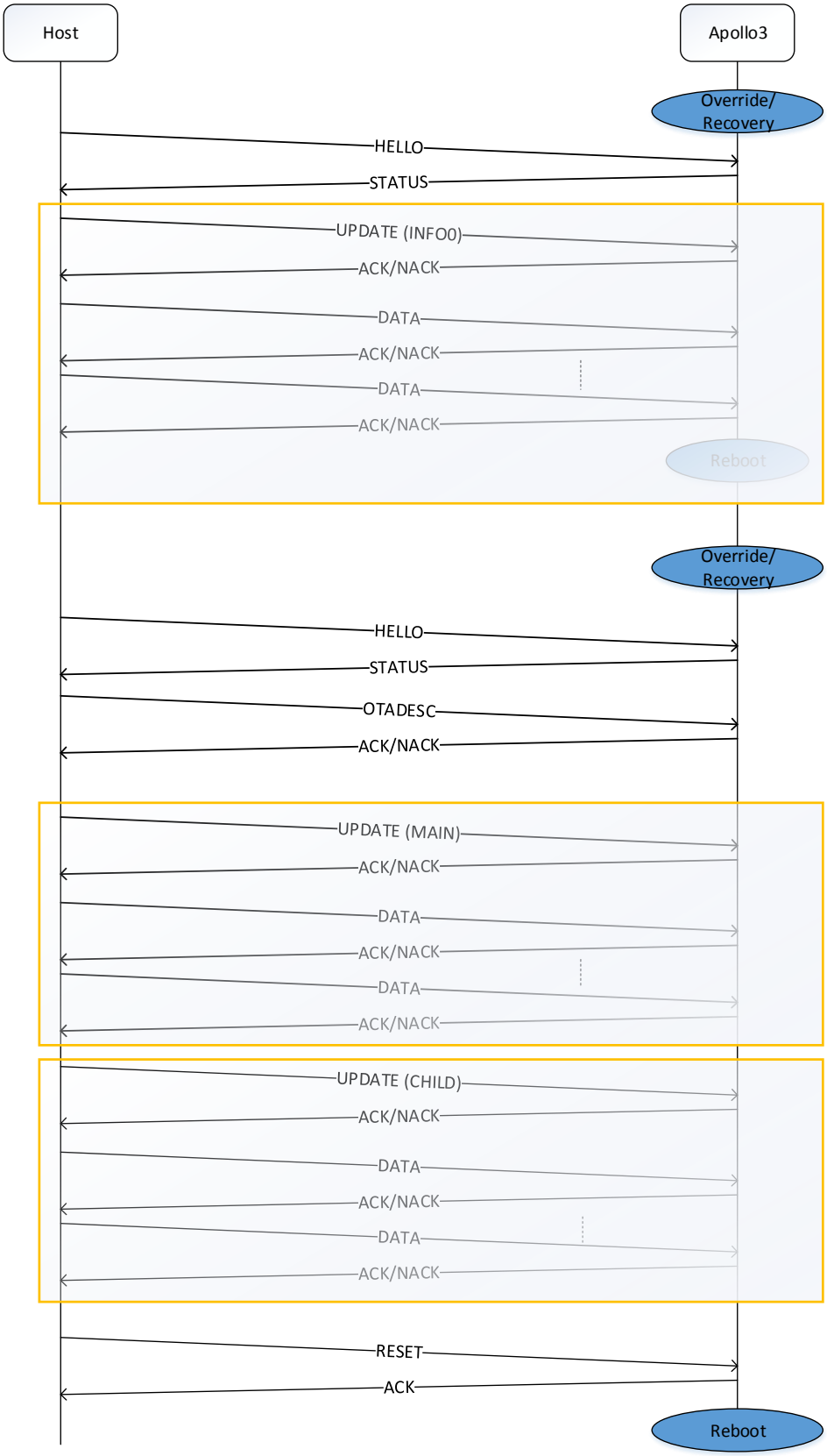


Figure 1 – UART Wired Update/Recovery Process

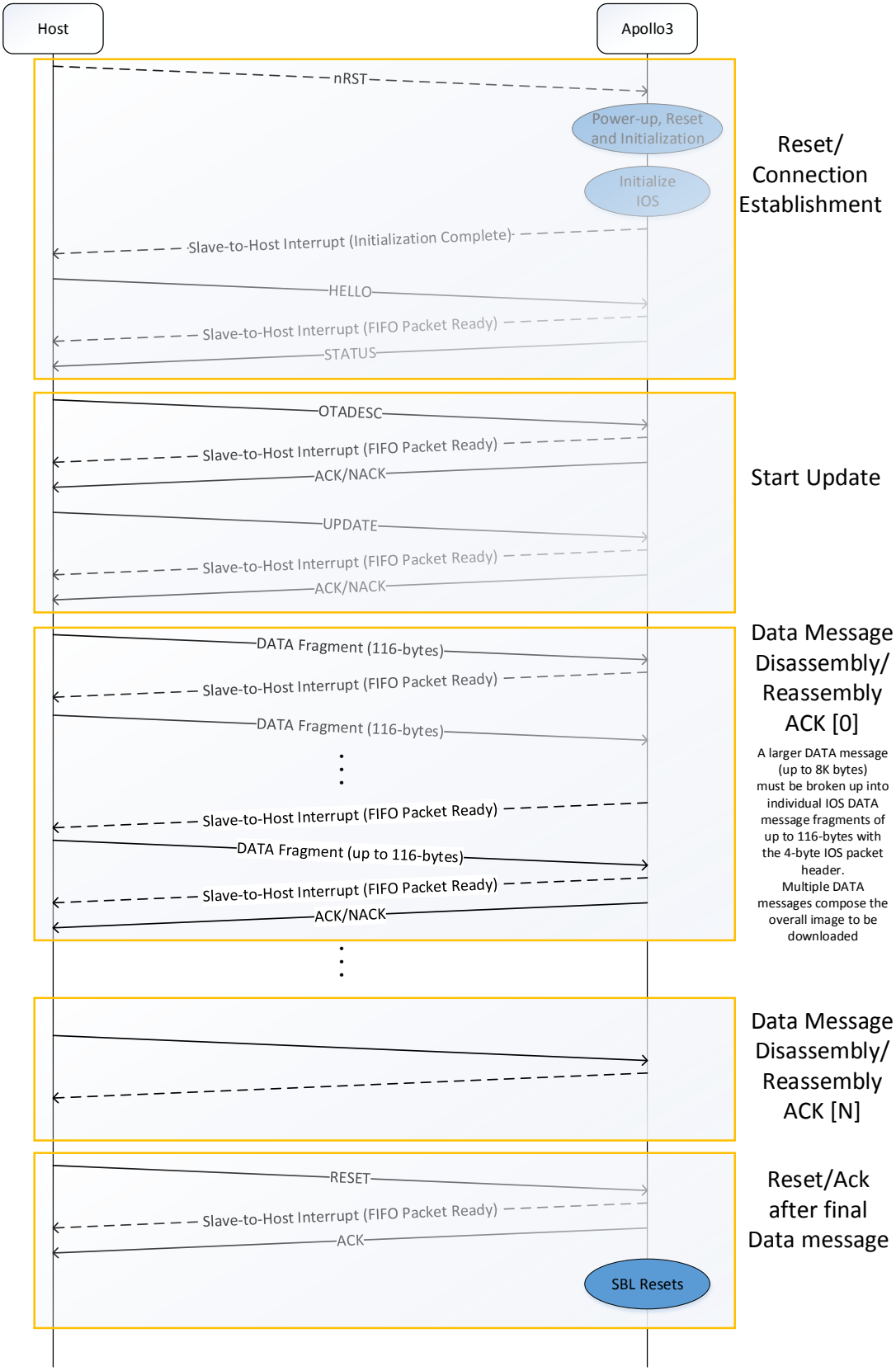


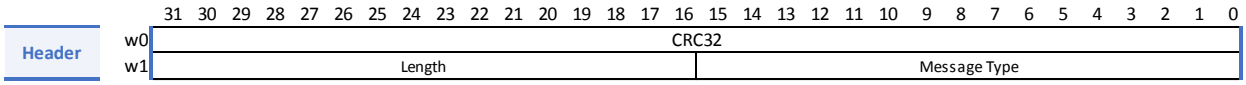
Figure 2 – IOS Wired Update/Recovery Process

6.1 Wired Update Messages

All the message formats below assume little-endian byte order.

Each message starts with a common header which defines the following message type and length, along with a CRC for error checking.

Common Header

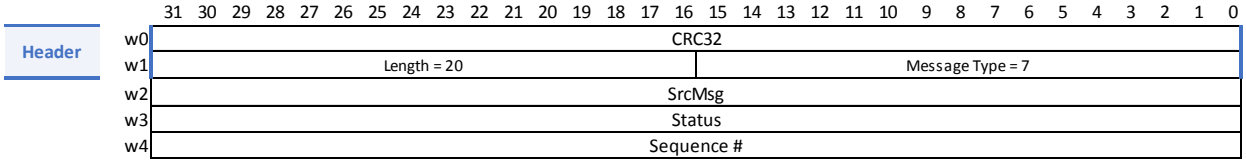


CRC32	CRC32 computed over the rest of message
Message Type	Message Type 0 = HELLO, 1 = STATUS, 2 = OTADESC, 3 = UPDATE, 4 = ABORT, 5 = RECOVER, 6 = RESET, 7 = ACK, 8 = DATA,
Length	Total Length of message (including header)

6.1.1 Acknowledgement (ACK) Message

The SBL acknowledges most of the messages received using an ACK message (except for RECOVER, which is not acknowledged unless encountering a failure, and HELLO, which is acknowledged using STATUS message). Acknowledgements for DATA messages (described later), also include a sequence number, which can be used to implement a retransmission mechanism at host side if the connection medium is lossy.

Ack Message

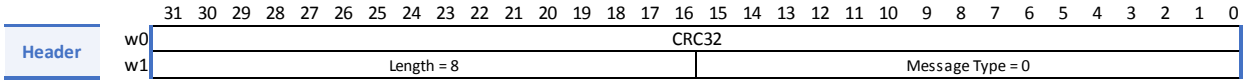


SrcMsg	Message type being acknowledged
Status	Status (ACK/NACK)
Sequence#	Next Sequence number expected (Only relevant for DATA ACK)

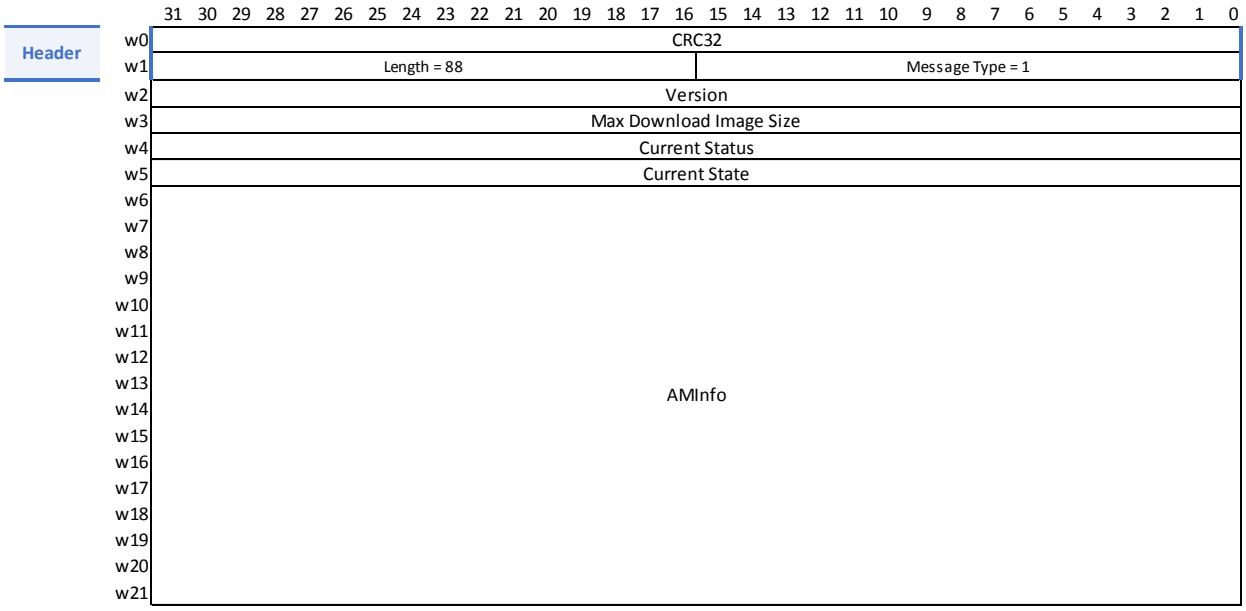
6.1.2 Connection Establishment (HELLO and STATUS) Messages

There is initial handshake between the host and SBL which can assist in determining the reason why SBL got into the wired update mode. A HELLO message is sent from host, to which the SBL responds with a STATUS message. The STATUS message also provides information about the largest size of the image blob that can be downloaded using SBL.

Hello Message



Status Message



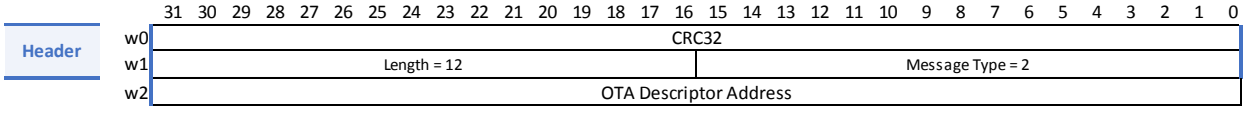
Version	Bootloader Version information
Max Image Size	Maximum size of blob that can be transferred using UPDATE command
Status	Current Boot Status
State	Current Bootloader state
AMInfo	Aux information - only for Ambiq Micro usage

6.1.3 Secure Upgrade (OTADESC) Message

Upgrade for the firmware images using wired update undergoes the same secure OTA flow as applicable to wireless OTA. SBL just provides means to do wired download for the image blobs. So, in accordance there are messages which instruct SBL to create an OTA Descriptor, followed by download of one or more image blobs (using UPDATE and DATA messages).

SBL reserves the complete flash page starting at the specified address for the purpose of building the OTA Descriptor.

OTA Descriptor Message



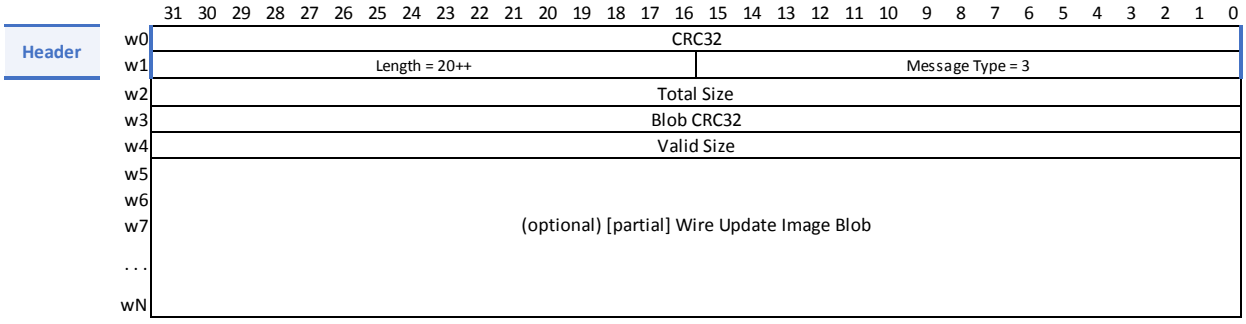
OTA Descriptor Address	Flash address where to build OTA Descriptor (must be page aligned)

6.1.4 Wired Download (UPDATE and DATA) Messages

Any upgrade on the device comprise of an UPDATE message which describes the nature and size of upgrade. It also contains the required security information for verification of the image. The UPDATE message is then followed by zero or more DATA messages to send the actual Image blob. After all the data is received, SBL verifies the integrity and validates it as per the prevalent security policy.

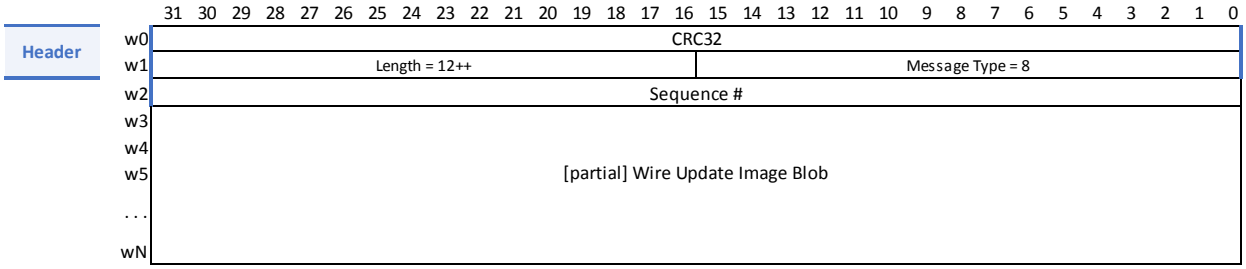
To avoid corrupting existing flash space with corrupted downloads, the image blobs are downloaded to SRAM, and written to flash only after verification for integrity & authentication. This implies that individual wired downloads are limited in size based on the available SRAM. Bigger size blobs can still be transferred by splitting them accordingly.

Update Message



Total Size	Total Size of Image Blob
Blob CRC32	CRC32 computed over Image Blob
Valid Size	Size of part of Image Blob in this message (optional)

Data Message

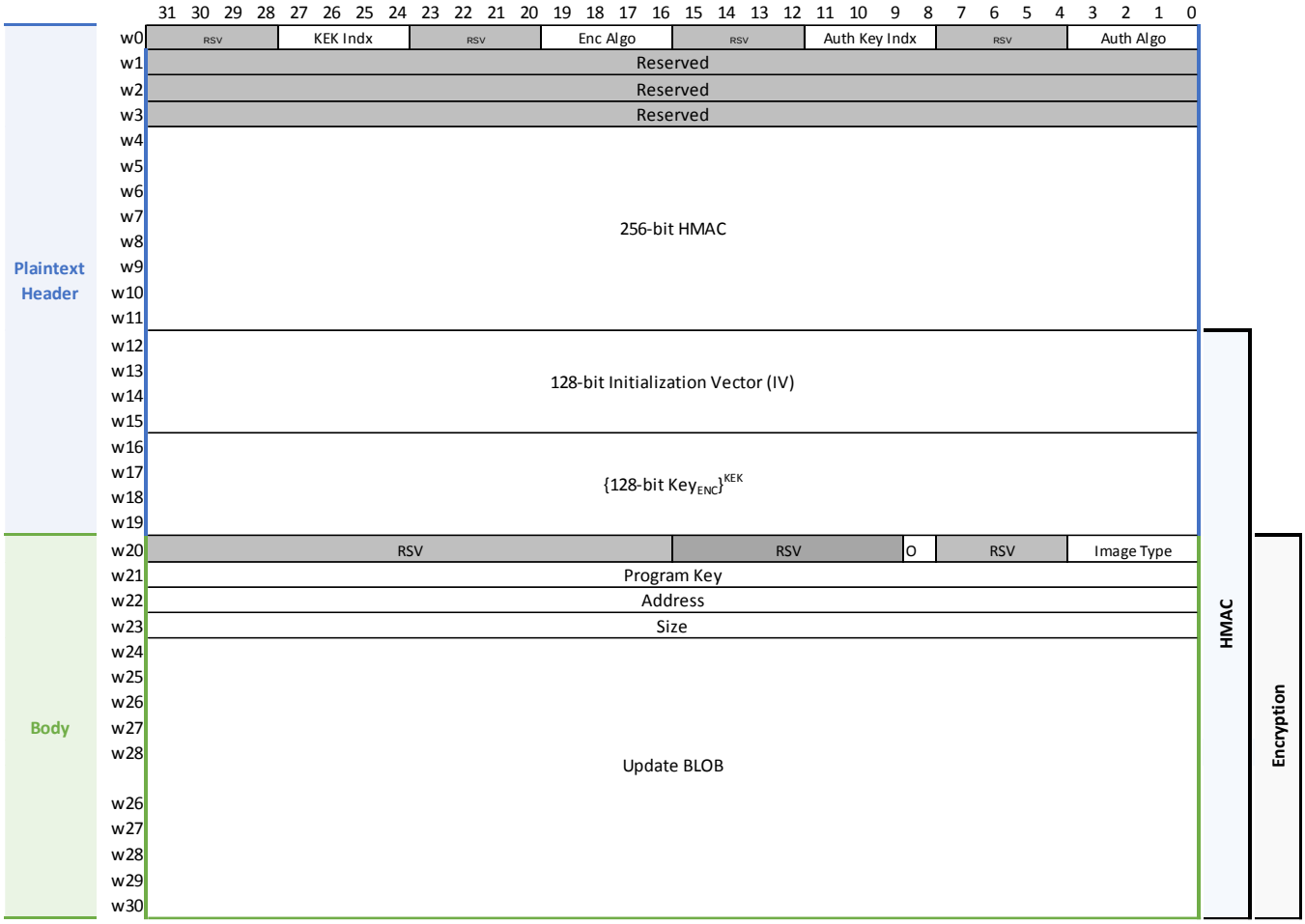


Sequence#	Sequence number - Byte offset within the Image Blob which corresponds to following Data

6.1.4.1 Image Transfer

The Image blob downloaded using UPDATE/DATA messages itself is constructed as below.

Wired Update Image Blob



KEK Indx	Key-Encryption-Key Index - this index specifies which KEK should be used within the KEK bank for all key unwrap functions
Enc Algo	Encryption Algorithm - specifies which algorithm is to be used for decrypting the image 0: N/A 1: AES-128 CBC others: not supported
Auth Key Indx	Authentication Key Index - this index specifies which authentication key should be used within the Auth Key bank for authentication of this image
Auth Algo	Authentication Algorithm - specifies which algorithm is to be used for authentication the image 0: N/A 1: SHA-256 others: not supported
256-bit HMAC	256 bit HMAC signature of the Install Blob following
128-bit IV	128-bit Initialization Vector used for seeding the encryption/decryption of the image
{128-bit Key_{ENC}}^{KEK}	128-bit KEK wrapped encryption key
Image Type	Identifies the type of image 0 = SBL, 1 = AM3P, 2 = PATCH, 3 = MAIN, 4 = CHILD, 5 = OTHER, 6 = NONSECURE, 7 = INFO0, 32 = INFO0-NO-OTA
O	Implying an OTA process request
Program Key	32-b key required for programming the update (INFO_KEY for INFO0-NO-OTA, PROG_KEY for all other image types)
Address	Address in flash to load the image to (For INFO0-NO-OTA update - this implies the offset in the Infospace where the update needs to happen in 4-byte multiples)
Size	Size of the Update Blob

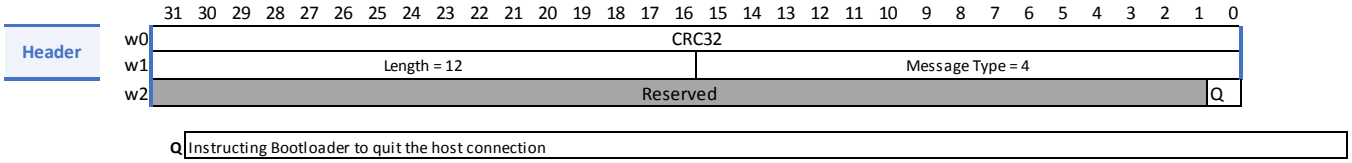
This Wired Download mechanism can be used to write some data to flash directly, direct program info0, or to download an image blob to the device for subsequent OTA by controlling the ‘O’ flag in the downloaded image blob header.

An INFO0 program (using ImageType = INFO0-NO-OTA) results in immediate Reboot of the device after the last DATA message is received and acknowledge.

6.1.5 Termination (ABORT) Message

A Download in progress can be aborted using ABORT message. Host has a choice to continue the connection, or instruct SBL to quit the connection.

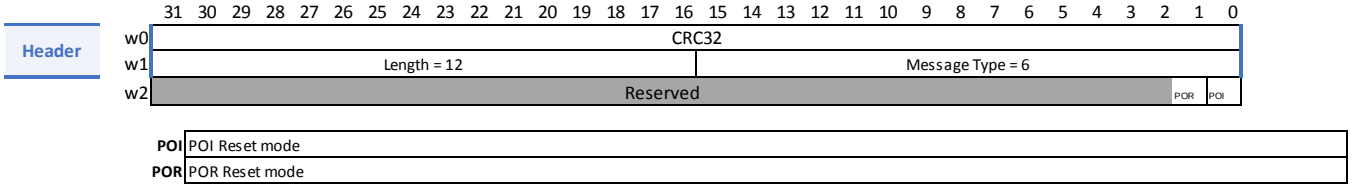
Abort Message



6.1.6 Reboot (RESET) Message

The Image blobs downloaded through UPDATE/DATA messages will have ‘O’ bit set to instruct SBL to schedule an OTA using the downloaded image. Multiple images can be combined together for the download step using this process. Actual Image upgrade is only initiated when a RESET message is received, as part of regular OTA processing by SBL.

Reset Message



6.1.7 Device Recovery (RECOVER) Message

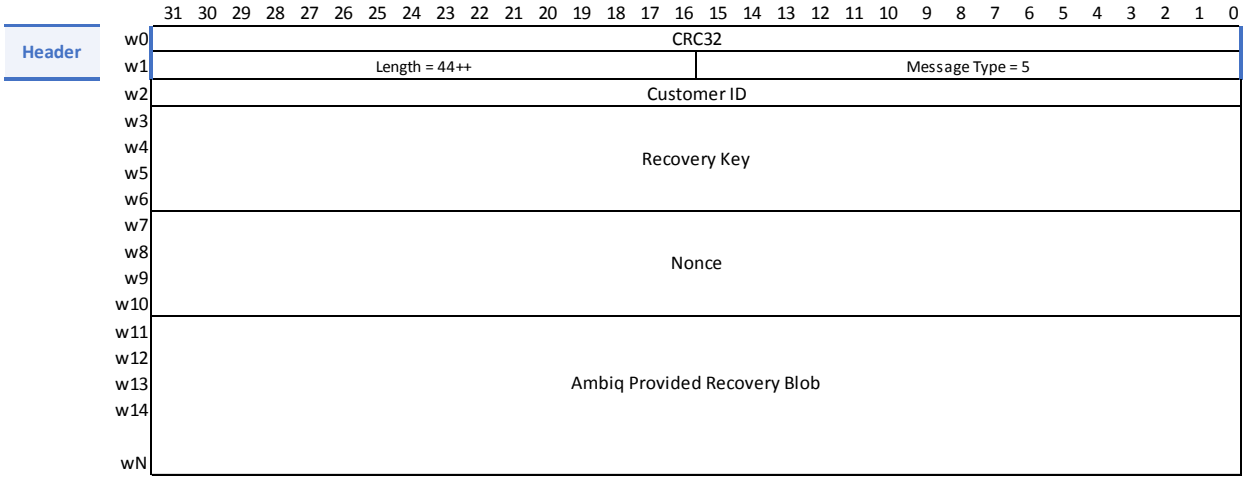
Device Recovery procedure allows the user to factory reset the device. Its usage is different depending on whether it is a secure or non-secure device.

A corrupt INFO0 (e.g. invalid signature, or invalid values for Security fields) on a secure SKU Apollo3-Blue MCU 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. There are multiple level of security put in place for this.

- To generate a RECOVER message with proper credentials, customer would need to contact Ambiq using a secure channel[2] and provide certain details (Unique CustomerID assigned to them, and a CHIP Part# along with a unique 128b Nonce value). Ambiq will then provide a “Recovery Key”, which is bound to the CustomerID, nonce and particular part.
- The “RECOVER” message contains this key along with customer supplied Nonce & CustomerID. The Recovery Key is authenticated by SBL using Ambiq Keys. Thereafter, only if all the parameters match – a factory reset is issued for the part, which erases INFO0 and the user flash.

[2] Implementation of the infrastructure to support secure recovery is a work in progress and details will be provided in future documentation.

Recover Message



Customer ID	Unique Customer ID for the user
Recovery Key	128b Recovery Key value
Nonce	128b Nonce Value - should correspond to the one given to Ambiq to generate the Recovery blob
Recovery Blob	Recovery Blob generated by Ambiq and provided to customer

7. Device Recovery Procedure

Ambiq provides means to factory-reset a device. Likely reason could be misconfiguration of INFO0, resulting in bricked secure part. Even for the non-secure parts, this procedure could be used to revert back to the factory settings.

7.1 Request for Ambiq Recovery Blob (needed for secure parts)

Customer would need to contact Ambiq using a secure channel (i.e. the Ambiq Security Portal – using their unique credentials) and provide certain details

- Unique customerID assigned to them
- a CHIP Part# (A contiguous Range is supported – if all owned by Amazon)
- The range will be validated against Ambiq records to ensure they were shipped to the customer (based on customer ID)
- A unique 128b nonce value

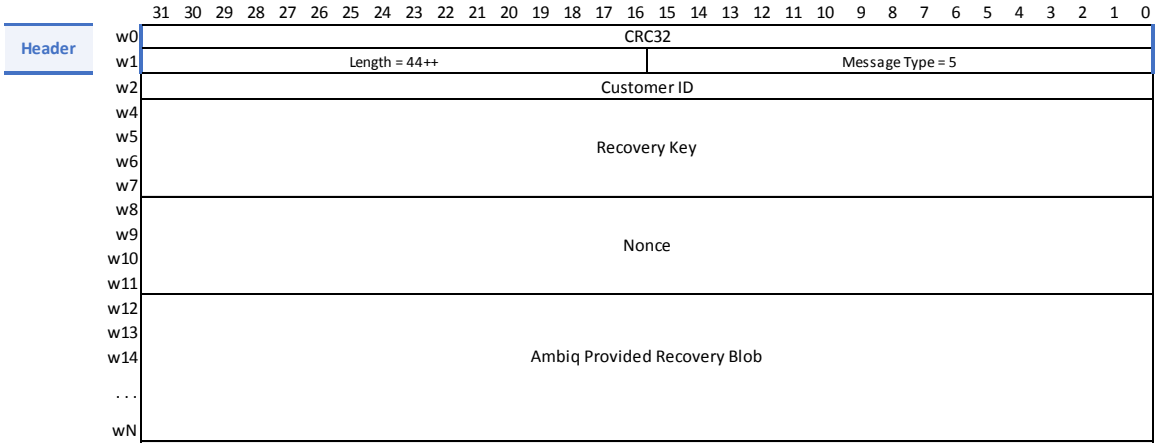
Once the provided information is validated, Ambiq generates a unique “Recovery Blob”, which is bound to the customerID, nonce and particular Part(s). The contents of the blob are encrypted and signed by Ambiq.

7.2 Factory Reset using RECOVER message

To factory reset a device, a wired host needs to send a special RECOVER message to SBL during the Wired Update phase, as below.

This message also contains the Ambiq provided Recovery Blob, as generated above through a separate channel.

Recover Message



Customer ID	Unique Customer ID for the user
Recovery Key	128b Recovery Key value
Nonce	128b Nonce Value - should correspond to the one given to Ambiq to generate the Recovery blob
Recovery Blob	Recovery Blob generated by Ambiq and provided to customer

8. Secondary Bootloader

Apollo3 Secure Boot/Update flow allows for provision to incorporate a secondary bootloader. A Secondary bootloader could be used to supplement the core capabilities of the native Apollo3 SBL. Potential reasoning for implementing one could be:

- Need to support different authentication/encryption algorithms
- Need to support external FLASH
- Other vendor specific enhancements

For designs incorporating a secondary bootloader, the latter replaces the main image. SBL treats the Secondary bootloader as the main image and verifies/updates the same using native boot/update flow. The secondary bootloader can then implement the additional features before passing control to the main firmware.

The native Secure Update flow can be leveraged to pass/return information about the proprietary OTA images to the secondary bootloader as well. All that is required is for the OTA image headers (as pointed to from the OTA Descriptor) to be residing in internal flash, and have the same structure for the first four bytes (specifically the magic number field). Images with the following magic numbers are considered proprietary image containers, and are passed on to the Secondary Bootloader for further processing transparently.

- Magic number passed transparently to secondary bootloader: 0xC1 – 0xCA, 0xCD, 0xCE

8.1 Device Programming Considerations for Secondary Bootloader

There are certain considerations while OEM programming a design with a secondary bootloader.

- Ensure INFO0_SECURITY. PLONEXIT is set to 0 when programming customer infospace
    - This ensures INFO0 space (including the key area) is accessible to be used for image verification purpose
    - It also allows Secondary bootloader to implement page lockouts for Read/Write protection by clearing additional bits in the register REG_MCU_CTRL_FLASH_WPROT* & REG_MCU_CTRL_FLASH_RPROT*

8.2 Secondary Bootloader programming considerations

8.2.1 OTA Processing

When using this extended Update flow using the special magic numbers, secondary bootloader undergoes the following processing sequence:

- Access the OTA Descriptor using the register REG_MCU_CTRL_OTAPOINTER.
- It processes the OTA images by scanning through the list with valid images marked as “Pending”.
    - Note: OTA images in external flash could be handled in Secondary Bootloader by building a dummy OTA header in internal flash which then contains pointer to actual image resident in external flash.
- It is the responsibility of the Secondary Bootloader to provide feedback for respective OTA images to the main firmware
    - OTA feedback – provided by clearing respective bit(s) in the OTA descriptor, as described in section 5.2
- “Invalidate” the OTA pointer once all the images are processed.
    - Clear bit OTAVALID in register REG_MCU_CTRL_OTAPOINTER

8.2.2 Asset Protections

It is the responsibility of the secondary bootloader code to do the following before transferring to the main image to ensure proper security.

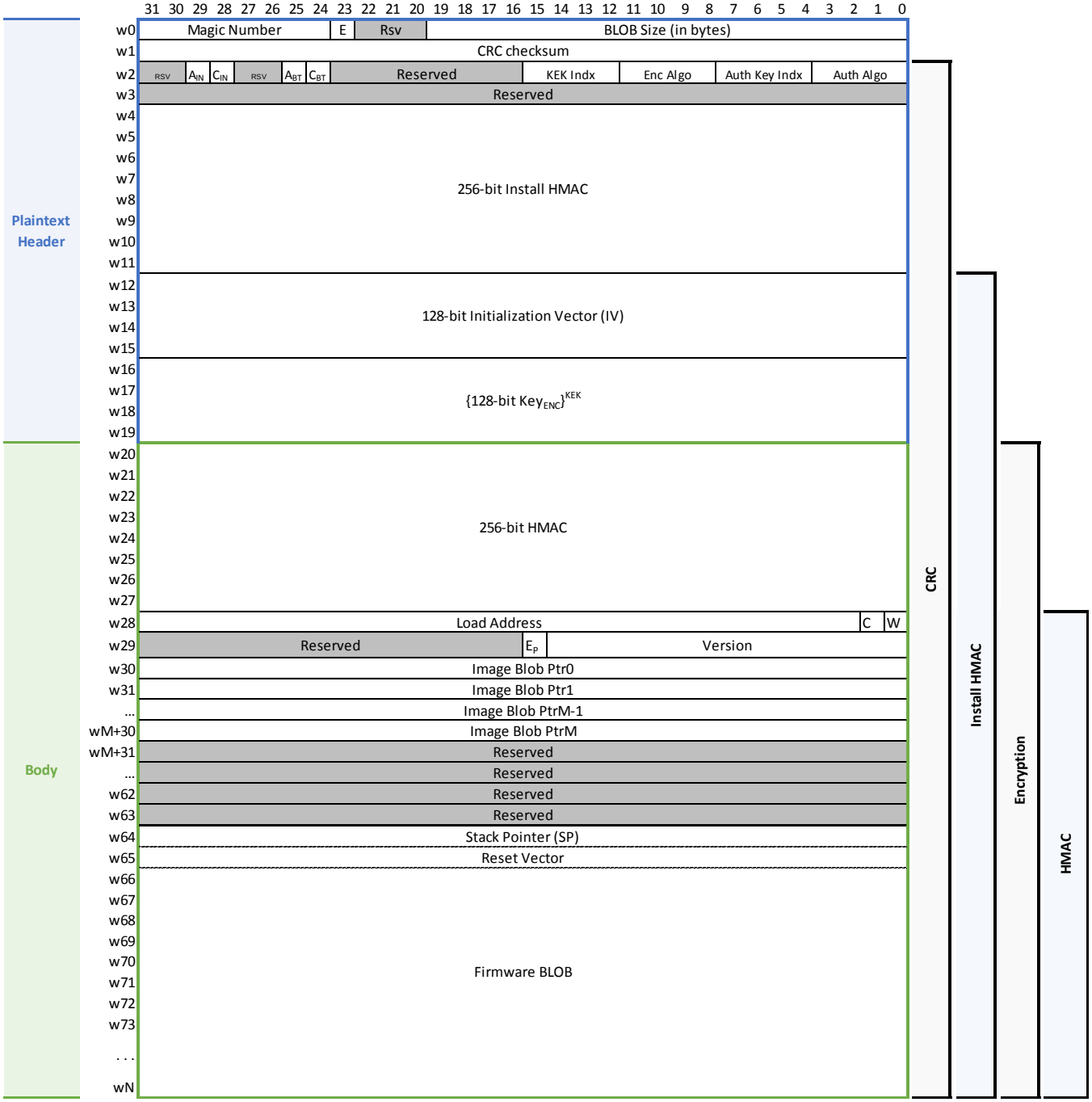
- “lock” the INFO0 space
    - This is accomplished by programming an invalid value (Anything other than the 128b customer key stored in infospace INFO0_CUSTOMER_KEY*) as customer key, e.g. by calling am_hal_security_set_key() with lockType as AM_HAL_SECURITY_LOCKTYPE_CUSTOMER)
- Restrict further access to flash protection registers
    - This is accomplished by asserting the protection lock by writing 1 to PROTLOCK bit in register REG_MCU_CTRL_BOOTLOADER

8.2.3 Debugger Support

There is one more consideration when implementing the Secondary Bootloader. If Debugger Support is disabled during the Secondary Bootloader phase (Controlled by SDBG bit in INFO0_SECURITY), Secondary Bootloader to implement additional logic to check if a halt is requested by the debugger after the bootloader, and if so halt the processor to allow a debugger to connect.

- Checking for a Halt request from Debugger – Check if Least Significant Bit of register REG_MCU_CTRL_SCRATCH0 is set
- If set, clear the bit & halt the processor using DHCSR register.

9. Sample OTA Processing for “Customer Main – Secure” Image



1. *Magic Number 0xC0 indicates it is “Customer Main – Secure” image*
2. *E (Encryption) & A_{IN} (Install time Authentication) values are verified against the minimum SECPOL configuration from INFO0:SECURITY (e.g. If SECPOL mandates Authentication, this bit MUST be set to 1, or else it is rejected)*
3. *A_{IN} Set indicates to SBL that the first step as Authentication of the Blob (possibly encrypted)*
   - *AuthKeyIdx is used to determine a Wrapped Key from the Key bank*
     - *The key is checked for revocation based on INFO0: AREVTRACK*
     - *KeyWrap configuration from INFO0:SECURITY is used to unwrap the wrapped key to extract the real Key to be used*

- *AuthAlgo is used to determine the Authentication algorithm (Only SHA256 HMAC supported currently)*
- *A signature is computed over the image w12 onwards – and verified against the signature in (256b Install HMAC <w4-w11>)*

4. *If "E" is set, the image is then decrypted*
   - *KEKIdx is used to determine a Wrapped Key Encryption Key (KEK) from the Key bank*
     - *The key is checked for revocation based on INFO0: KREVTRACK*
     - *KeyWrap configuration from INFO0:SECURITY is used to unwrap the wrapped key to extract the real KEK to be used*
   - *EncAlgo is used to determine the Authentication algorithm (AES128-CBC supported currently)*
   - *KEK is used to decrypt the Encrypted Encryption Key (w16-w19)*
   - *This Encryption key is then used to decrypt the encrypted blob (w20 onwards) using the IV from w12-w15*
5. *A_{BT} Set indicates to SBL to Authentication the image inside the encryption blob*
   - *AuthKeyIdx is used to determine a Key from the Key bank*
     - *The key is checked for revocation based on INFO0: AREVTRACK*
     - *KeyWrap configuration from INFO0:SECURITY is used to unwrap the wrapped key to extract the real Key to be used*
   - *AuthAlgo is used to determine the Authentication algorithm (Only SHA256 HMAC supported currently)*
   - *A signature is computed over the image w28 onwards – and verified against the signature in (256b Install HMAC <w20-w27>)*

6. *If "C_{IN}" is set, a CRC32 is then computed on the decrypted image (w2 onwards) and matched against CRC Checksum (w1)*
7. *Secure Bootloader can also check for validity of third party libraries before installing the main image, if such dependencies are called for in the main image header (Image Blob Ptr* - w34 onwards).*
8. *The image (full blob including the header) is "installed" at the address specified by load address (w28)*
9. *Bits "C" and "W" in w28 can be set to instruct SBL to Read and/or write protect the image after installation.*

Contact Information

Address	Ambiq Micro, Inc. 6500 River Place Blvd. Building 7, Suite 200 Austin, TX 78730
Phone	+1 (512) 879-2850
Website	http://www.ambiqmicro.com
General Information	info@ambiqmicro.com
Sales	sales@ambiqmicro.com
Technical Support	support@ambiqmicro.com

Legal Information and Disclaimers