"Secure Boot" SDK user guide

About this document

Scope and purpose

This document serves as a guide for using the “Secure Boot” SDK. The document explains about the SDK overview, software installation, provisioning flow, and “CySecureTools” design.

Intended audience

The “Secure Boot” SDK is intended for the users of the PSoC™ 64 devices.

Table of contents

1 Introduction

Infineon provides the “Secure Boot” SDK to simplify using the PSoC™ 64 “Secure Boot MCU” line of devices. This SDK includes all required libraries, tools, and sample code to provision and develop applications for PSoC™ 64 devices.

The “Secure Boot” SDK provides provisioning scripts with sample keys and policies, a pre-built Cypress Bootloader image, and post-build tools for signing firmware images. It uses the Python programming language.

1.1 Where to Get the “Secure Boot” SDK

The main component of the SDK is a Python package called “CySecureTools.” It is available for download here:

https://github.com/cypresssemiconductorco/cysecuretools

There are other components as well, and they are described in later sections of this document. For Windows users, all tools including “CySecureTools” are provided when installing ModusToolbox™ 2.3 or later.

1.2 Using this guide

This guide provides a high-level overview of the “Secure Boot” SDK, including details on how the provisioning process works, as well as descriptions of the provided scripts and tools. In addition, this guide provides a reference to the tokens/JSON structures used in the SDK.

This guide assumes you are familiar with the concept of public key cryptography, public/private key pairs, and digital signatures. An overview of these ideas is available here:

https://en.wikipedia.org/wiki/Public-key_cryptography

1.3 Definition of Terms

- **Root-of-Trust (RoT):** This is an immutable process or identity used as the first entity in a trust chain. No ancestor entity can provide a trustable attestation (in digest or other form) for the initial code and data state of the RoT.
- **Hardware Security Module (HSM):** A physical computing device that safeguards and manages digital keys for strong authentication, and that provides cryptographic processing. In the context of the PSoC™ 64 “Secure Boot MCU”, the HSM is a device programming engine placed in a physically secure facility.
- **Provisioning:** The process by which keys, policies and secrets are injected into the device. Once provisioned, the device can be accessed or modified only with the keys injected adhering to the relevant policies.
- **JSON:** JavaScript Object Notation (JSON) is an open-standard file format that uses human-readable text to transmit data objects consisting of attribute–value pairs and array data types (or any other serializable value).
- **JWT:** JSON Web Token (JWT) is an open, industry standard RFC 7519 method to securely represent claims between two parties.
- **JWK:** JSON Web Key (JWK) is a RF7517 compliant data structure that represents a cryptographic key.
- **Policies:** Policies are a collection of pre-defined (name, value) pairs that describe what is and is not allowed on the device. Most policies are enforced during boot-time by the RoT firmware in the device, some can be interpreted and enforced by higher layers of software like Infineon Bootloader.
- **“Secure Boot”:** Refers to a bootup process where the firmware being run by the chip is trusted by using strong cryptographic schemes and an immutable RoT.

- **Immutable Boot Code:** Refers to the first piece of code which is run after chip power-on before any user application is run. In the context of the PSoC™ 6 MCU family, it refers to the ROM and Flash code which is programmed at Infineon manufacturing and made immutable by transitioning life-cycle stages.
- **SWD:** Single Wire Debug, a two-wire debug port defined for Arm® Cortex® CPU's.
- **CMSIS-DAP:** CMSIS-DAP is a specification and a implementation of a Firmware that supports access to the CoreSight Debug Access Port (DAP).
- **DAPLink:** Arm Mbed DAPLink is an open-source software project that enables programming and debugging application software running on Arm® Cortex® CPUs.
- **KitProg3:** This is Infineon' low-level communication firmware for programming and debugging. It runs on a PSoC™ 5LP device. It is a is a multi-functional system, which uses SWD for programming and debugging, and provides a USB-I2C bridge, and USB-UART bridge. It supports CMSIS-DAP and DAPLink.
- **Single/Multi-image:** There are two different types of images that may be created, which define how the CM0+ "secure" co-processor and CM4 binaries are generated: Single-image and Multi-image. In Single-image mode, the CM0+ "secure" co-processor code binary is attached to the beginning of the CM4 binary to form a single binary. With the Single-image, the CM0+ "secure" co-processor and CM4 must be updated at the same time, requiring a single update binary. In Multi-image mode, the CM0+ "secure" co-processor and CM4 binaries are separate and therefore can be updated individually with two different update binaries. The default is Single-image mode, since few customers need to modify the secure CM0+ "secure" co-processor code base.
- **SMIF:** Serial Memory interface. In the context of this user guide, it refers to the highspeed Quad-SPI interface on PSoC™ 6 MCU.
- **Rollback Counter:** Special counter accessed by secure boot code that holds the value of the latest valid image and used in an anti-rollback protection mechanism. The goal of anti-rollback protection is to prevent downgrading the device to an older version of its software that has been deprecated due to security concerns.

2 Overview

The PSoC™ 64 “Secure Boot MCU” line, based on the PSoC™ 6 MCU platform, features out-of-box security functionality. The line provides an isolated RoT with true attestation and provisioning services. In addition, these MCUs deliver a pre-configured secure execution environment which supports system software for various IoT platforms and provides:

- Secure provisioning
- Secure storage
- Secure firmware management

To develop with a PSoC™ 64 “Secure Boot MCU”, you first provision the device with keys and policies. You then program the device with signed firmware. Otherwise the device will not boot up correctly. The “Secure Boot” SDK provides development tools to demonstrate the provisioning and signing flow.

In addition, Infineon Bootloader enables “Secure Boot” and Firmware updates.

2.1 Secure Boot SDK components

The “Secure Boot” SDK is organized as a stand-alone python “CySecureTools” package, which contains all the required scripts, default provisioning packets, and the default policy file, as follows:

Table 1 Component and its Purpose details

Component	Purpose
Command line tool	Allows using “CySecureTools” as command-line utility to perform all required operations.
Provisioning Scripts	Python scripts for provisioning the PSoC™ 64 “Secure Boot MCU”. Scripts are based on Python.
Entrance Exam Scripts	Runs an entrance exam on the PSoC™ 64 “Secure Boot MCU” to ensure no tampering has occurred.
Infineon Bootloader Image	The first stage bootloader based on an open source MCUBoot[1] library.
Sample Provisioning Policies	Examples to be used as templates for forming provisioning tokens.

[1] https://mcuboot.com/

2.2 How does the “Secure Boot” SDK work?

The goal for a developer creating a design using a secure device is to ensure that the software running on it is authorized and unchanged. The “CySecureTools” package provides the tools you use to make that happen. This section describes what you do and how it works at the highest level. Subsequent sections in this user guide provide details.

“CySecureTools” provides a reference implementation for the patent-pending process developed by Infineon for securing the software on a device. It is based on industry-best practices in public-key infrastructure (PKI), cryptography, and digital signing. From factory to bootup to remote updates, every step in the process is signed and verified.

Every secured MCU is embedded with a Infineon-owned Root of Trust (RoT) when it comes out of the factory. This RoT is based on a public key that is owned by Infineon.

When an OEM customer purchases device, there is a secured process to transfer the RoT to the OEM. This process replaces the Infineon public key with the OEM public key. From that point on, the OEM “owns” the device and the secured device will only accept any further security-related interaction if it is appropriately signed by the OEM.

For “Secure Boot” functionality, Infineon provides immutable boot code programmed on factory that launches a Infineon Bootloader. That bootloader is itself signed, so that the boot code can verify the integrity of the bootloader. This bootloader then verifies the OEM application firmware that should run on the device before launching that code. Infineon provides a secured bootloader as part of “CySecureTools”, but a customer can use their own.

To work with secure devices, the OEM provides three “things”:

- A set of cryptographic keys, the public key of which will be used for validating OEM application firmware.
- A set of security policies that define how the secure chip should behave.
- Certificates (optional) used to bind device identity or provide a chain-of-trust to a higher certifying authority.

This information, along with the bootloader, is securely injected into the device before firmware is programmed. This process is called ***provisioning***. This information provides the rules that the boot code follows when launching the bootloader, and provides the resources required to verify the authenticity of the code.

The OEM develops the software to run on the device and digitally signs the application using a private key that corresponds to the keys provisioned in the device. With policies and keys in place, the boot code verifies the bootloader, which verifies the signature and the integrity of any code before launching it.

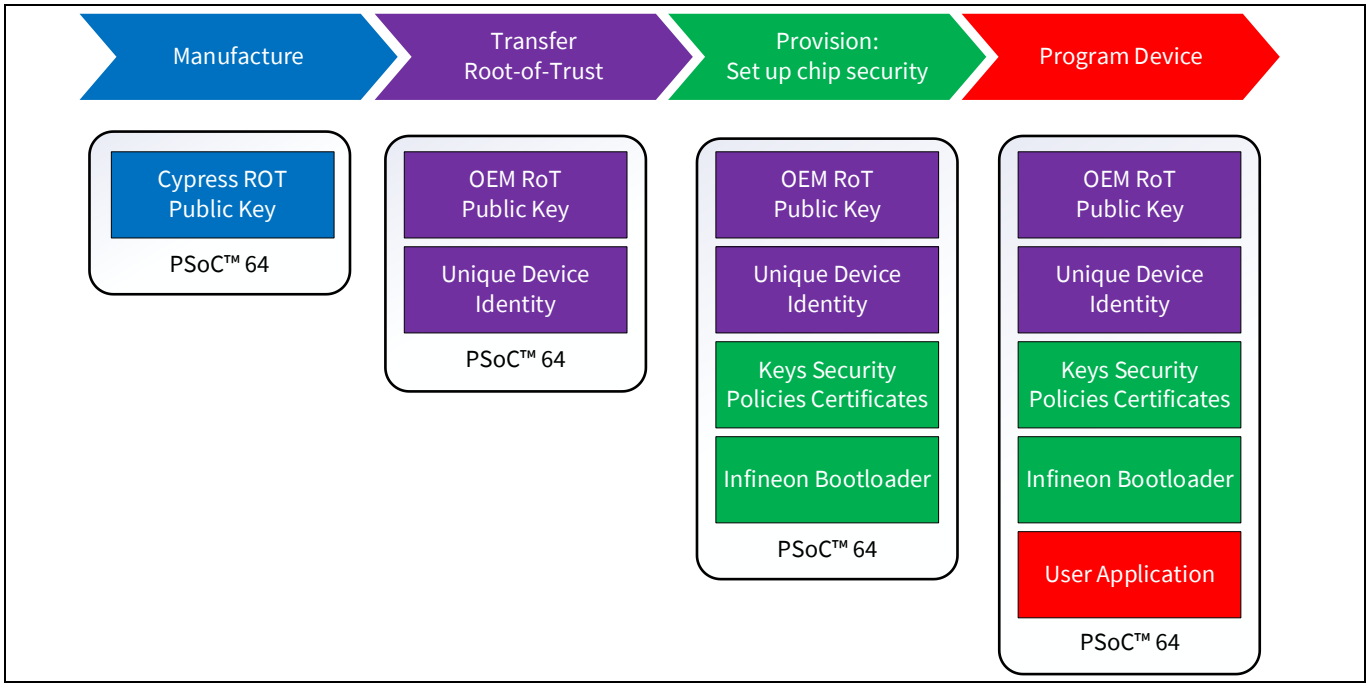

Figure 1 PSoC™ 64 usage processes

"CySecureTools" includes:

- Provisioning scripts (in Python) that provide an API for tasks such as
  - Creating the required keys
  - Specifying security policies and debug access
  - Provisioning the device
  - Forming device identity certificate
  - Enabling secure debug
- The Infineon Bootloader (as a binary image with an associated certificate)
- An optional entrance exam step to ensure device integrity
- Sample provisioning policies

2.3 What is Provisioning?

Provisioning is a process whereby secured assets like keys and security policies are injected into the device. During development, a software team can manage this. During production, this step typically occurs in a secure manufacturing environment that has a Hardware Security Module (HSM). For the PSoC™ 64 "Secure Boot MCU", provisioning involves the following steps:

- Transferring the RoT from Infineon to the development user (called OEM in this document).
- Injecting user assets such as image-signing keys, device security policies, and certificates into the device.

2.3.1 Transferring RoT

Every PSoC™ 64 "Secure Boot MCU" has a Infineon public key placed in the part during manufacturing. This Infineon public key acts as the initial RoT for the device after it is manufactured.

The RoT transfer process can be represented as a series of trust claims; exchanged between the following entities:

- Infineon – The owner of the Infineon Root private key.
- Secure Manufacturing environment HSM – The entity authorized to provision and program the PSoC™ 64 "Secure Boot MCU".
- OEM/Developer – The user/code developer of the part.
- PSoC™ 64 "Secure Boot MCU" – The holder of the Infineon Root public key.

The following illustration shows a high-level view:

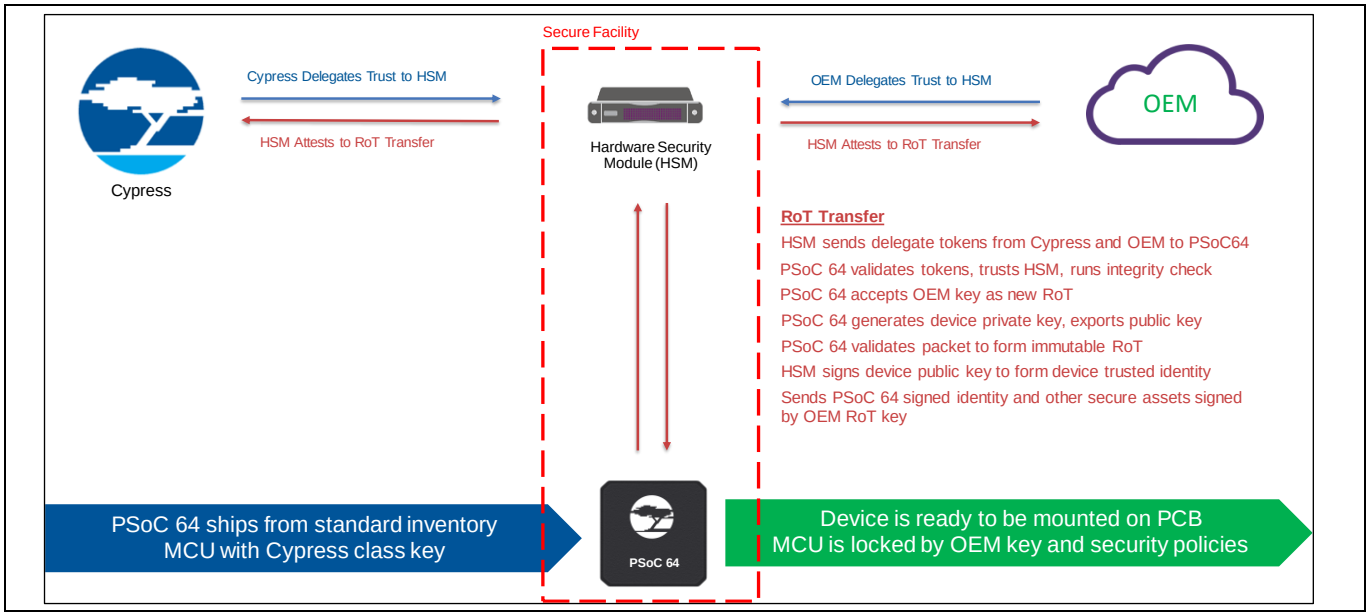

Figure 2 Infineon authoring the HSM

The series of steps to transfer the root-of-trust include:

1. Infineon authorizes the HSM to provision a part.
2. The OEM/User authorizes the same HSM to provision the part with credentials and firmware.
3. The HSM then presents the above authorization objects to the PSoC™ 64 "Secure Boot MCU."
4. The PSoC™ 64 "Secure Boot MCU" verifies authorization signatures and claims. If all are valid, the chip accepts the OEM RoT public key and allows the HSM to send provisioning packets.

The end result of this RoT transfer process can be represented as follows:

- The PSoC™ 64 "Secure Boot MCU" now uses the OEM RoT public key as the root key used to validate any OEM asset (image keys, policies etc.). This permanently and irrevocably replaces the Infineon RoT.
- The PSoC™ 64 "Secure Boot MCU" now trusts the HSM public key and expects all further provisioning packets to be signed by the corresponding HSM private key.

The actual authorization objects for the PSoC™ 64 "Secure Boot MCU" are represented using the JSON Web Token (JWT) format. A simplified view of the flow of the Infineon and the OEM authorizing a HSM is shown in the following diagram:

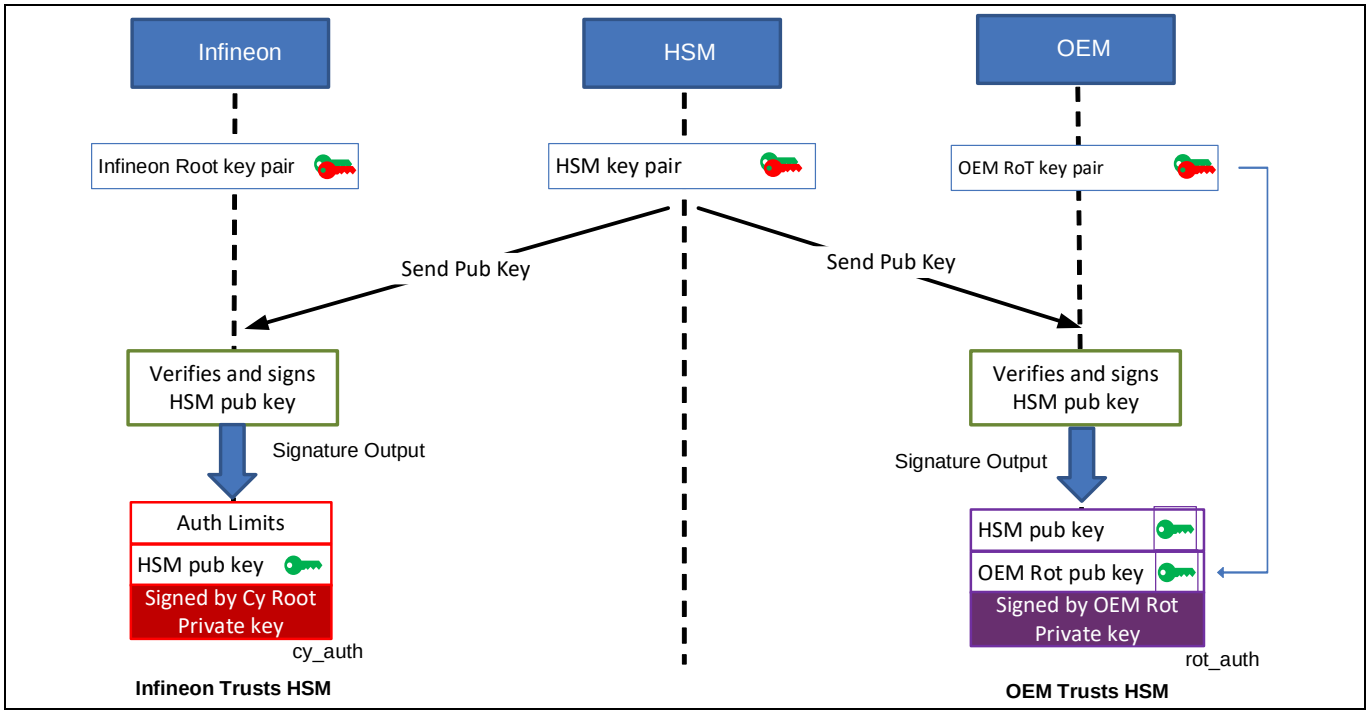

Figure 3 OEM/User authoring the HSM

The final output of this process generates the following JWTs:

- **cy_auth.JWT**: Contains the public key of the HSM to be trusted. Additional fields such as an expiration date can be specified to limit this token's use.
- **rot_auth.JWT**: Contains the public key of the HSM to be trusted as well as the OEM RoT public key to which the RoT must be transferred.

The HSM then presents these tokens to the chip, as shown in the following diagram:

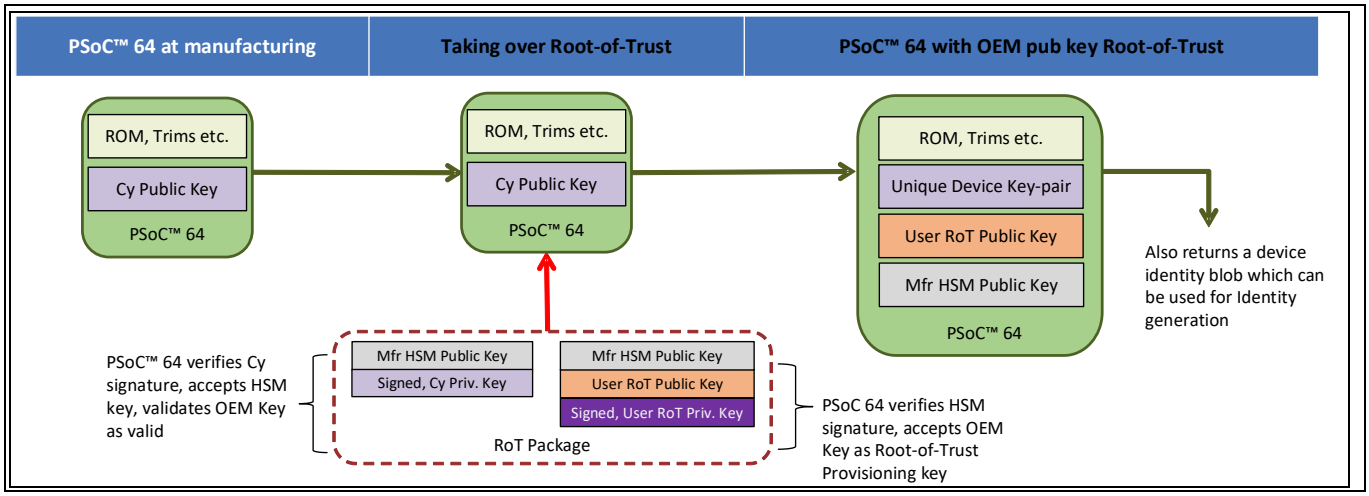

Figure 4 HSM presenting the authorization to PSoC™ 64

After the root-of-trust packet is sent, the device also generates a unique device key pair and exports the generated device public key and its unique ID. This combination can be used to chain the identity of the chip to a Certifying Authority trusted by the OEM.

2.3.2 Injecting User Assets

After the RoT is transferred to the OEM RoT public key, the user can inject several assets into the device. These include:

- Public Keys
  - Image public key – Used by the bootloader to check the next image signature.
- Device Policies
  - Boot & Upgrade policy – Specifies which regions of flash constitute a bootloader and launch image, as well as the key associated when validating the flash area.
  - Debug policy – Specifies the behavior of the device debug ports (CM0+ “secure” co-processor/CM4/SYSAP). Also, specifies the device behavior when transitioning into RMA mode.
- Chain-of-Trust Certificates
  - Any certificates needed on the device; for example, device certificate for TLS or Identity.

Both public keys and device policies are present in a JWT token called ‘prov_req.JWT.’ They are signed by the OEM RoT private key.

The certificates present in the chain-of-trust may be signed by the same key, but no restrictions are placed on this field’s contents and the chain-of-trust is considered an opaque object.

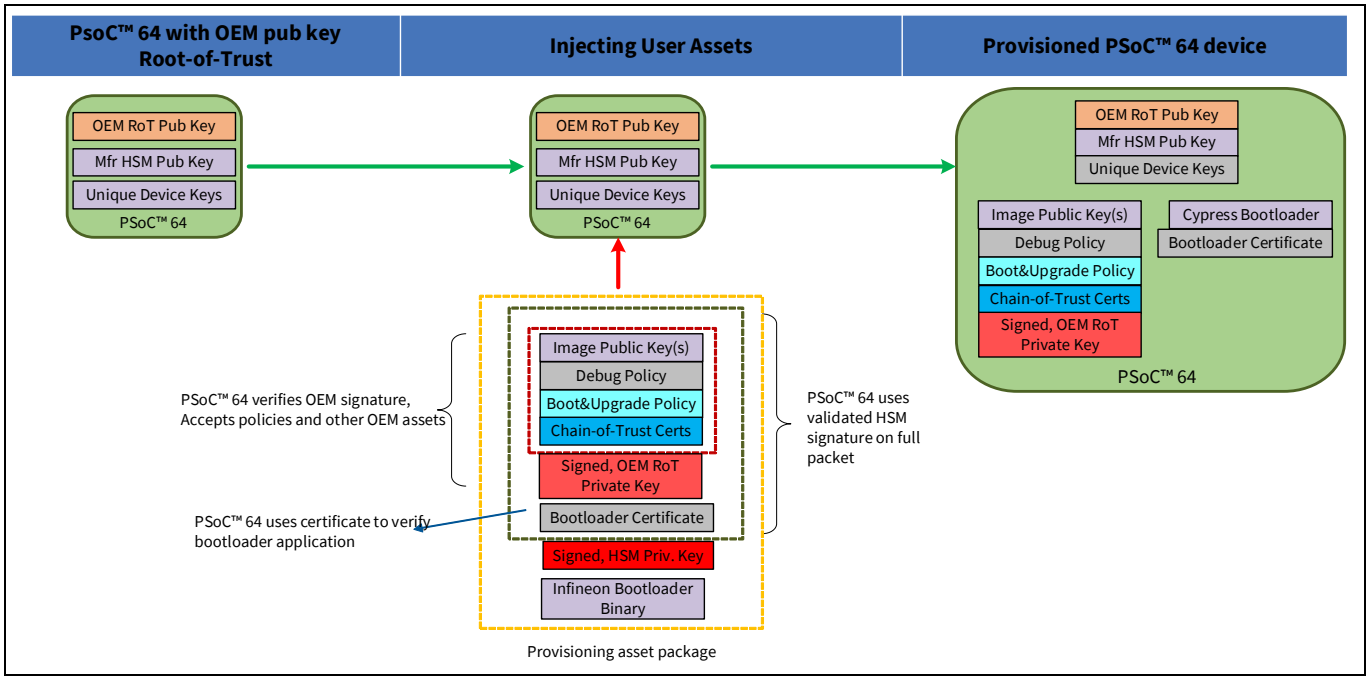

Figure 5 Provisioning flow

In addition to the OEM assets, Infineon Bootloader is programmed at this stage, along with the Bootloader Certificate (called 'image_cert.JWT') that has the signature of the Infineon Bootloader binary. This ensures that the bootloader itself can be verified and trusted. For more details on Infineon Bootloader, see the Cypress Bootloader section.

For more details on the exact provisioning packets, see the Provisioning script flow details section.

2.3.3 Re-provisioning User Assets

The PSoC™ 64 "Secure Boot MCUs" also allow some user assets to be re-provisioned, if allowed by the initial policy provisioned into the device.

The following assets are allowed to be re-provisioned

- Infineon Bootloader
- Public keys, Policies and Chain-of-trust certificate

All other assets such as the OEM RoT public key, HSM public key and device unique keys cannot be replaced using re-provisioning.

2.4 Infineon Bootloader

The Infineon Bootloader is included as a pre-built hex image. This image acts as the first image securely launched by the PSoC™ 64 "Secure Boot MCU" boot code. The Infineon Bootloader is based on an open source library MCUBoot and is capable of parsing the provisioned Boot&Upgrade policy and launch next image if all required checks pass. For more details about this open source library, refer to the MCUBoot Bootloader design website.

The bootloader recognizes two memory areas for each application partition. There is the Primary Slot and a Secondary Slot. The Primary Slot contains the Boot Image, and the Secondary Slot contains the Upgrade Image. The Boot Image is the application code that executes. The Upgrade Image is where the code upgrade is stored before it is copied to the Boot Image. Code cannot execute in the Secondary Slot. In the Single-image mode, there is one Primary Slot and one Secondary Slot since the CM0+ "secure" co-processor and CM4 binaries are combined. In the Multi-image mode there are four slots, two for the CM0+ "secure" co-processor (Primary and Secondary) and two for the CM4 (Primary and Secondary), which allows the CM4 and the CM0+ "secure" co-processor code to be upgraded individually if needed.

There are two methods supported for firmware upgrades, Replace and Swap. The Replace method simply copies the Secondary Slot into the Primary Slot, then invalidates the Secondary Slot. The Swap method safely swaps the Primary and Secondary slots so that you may revert to the previous version of firmware. After the Primary Slot has been updated and validated in either method, the new firmware is executed.

Note The current version of Infineon Bootloader supports image Replace upgrades in the 512K and 1M flash devices and image Swap for the 2M flash devices. The Swap mode requires additional 64K of internal flash if both Primary and Secondary slots are on chip. If the Secondary slot is external, then only an additional 32K of internal flash is used.

The Infineon Bootloader supports external memory over the PSoC™ 64 Serial Memory Interface (SMIF). The bootloader currently only supports external memory vendors who support the Serial Flash Discovery Protocol (SFDP).

The Infineon Bootloader is capable of independently managing up to two user images for use cases where the Secure Processing Environment (SPE) code such as Trusted Firmware-M and Non-Secure Processing (NSPE) code needs to be independently updated with individual Boot and Upgrade slots.

The Infineon Bootloader also enforces the protection contexts for the bootloader code, so code running on another protection context cannot overwrite/tamper with the boot code. The following diagram shows the launch code sequence of Infineon Bootloader:

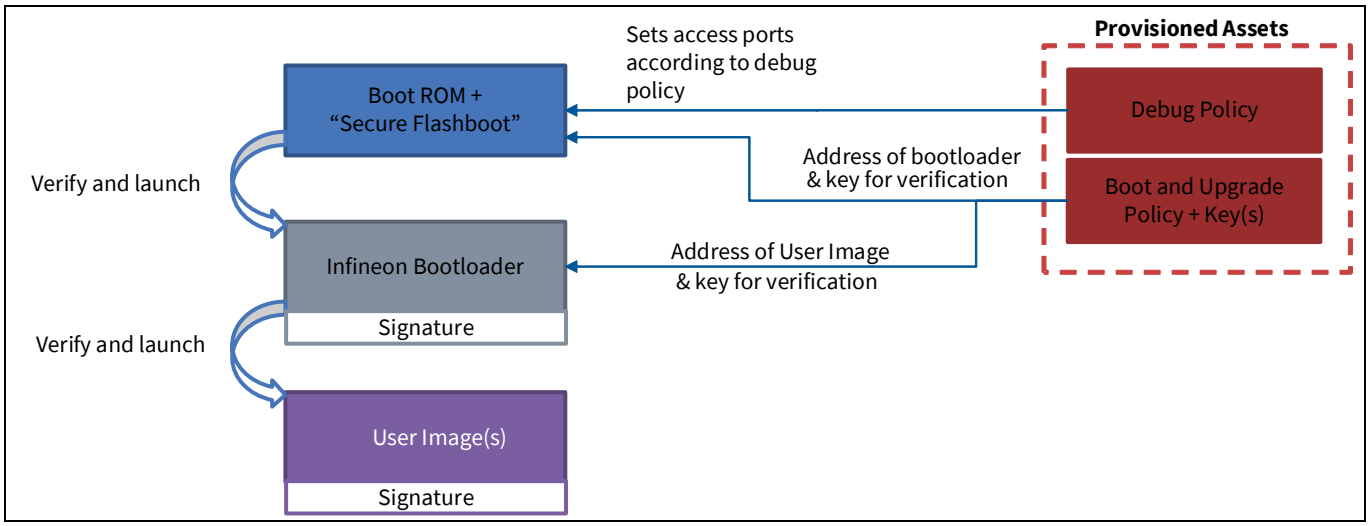

Figure 6 Bootloader launch sequence

During a normal bootup, the Infineon Bootloader performs the following operations:

- Reads the policies and parses them for further use.
- Checks if the upgrade slot is located in external memory and performs SMIF initialization correspondingly.
- Checks if the Boot Area (Primary Slot) contains an image to boot.
- Verifies that this image has the valid format.
- Verifies the image's digital signature.
- Verifies that the image rollback counter is greater than or equal to the value saved in the rollback protection counter of the boot code data.
- Checks if the Staging Area (Secondary Slot) has an image for upgrade.
- Boots Primary Slot if no correct image is found in the Staging Area.

If Staging area (Secondary Slot) has a new image, the Infineon Bootloader performs the following operations:

- Verifies the digital signature of the image located in Secondary Slot.
- Decrypts the image's body and verifies the digital signature of the decrypted image (optional for the encrypted image support).
- Checks that the corresponding policies allow upgrade.
- Checks that the image metadata matches the image in Primary Slot, then upgrades it.
- Replace Mode
  - Overwrites Primary Slot with the decrypted (if needed) Secondary Slot image.
  - Invalidates Secondary Slot by erasing the header and trailer (hash and signature) sections, so that at the next reset, the Secondary Slot is ignored.
- Swap Mode
  - Swaps the Primary Slot with the Secondary Slot so that the previous version may be recovered if there is a software issue.
  - The Secondary Slot is re-encrypted if it had been encrypted previously.

Overview

The following diagram shows a typical application update scenario using the Infineon Bootloader:

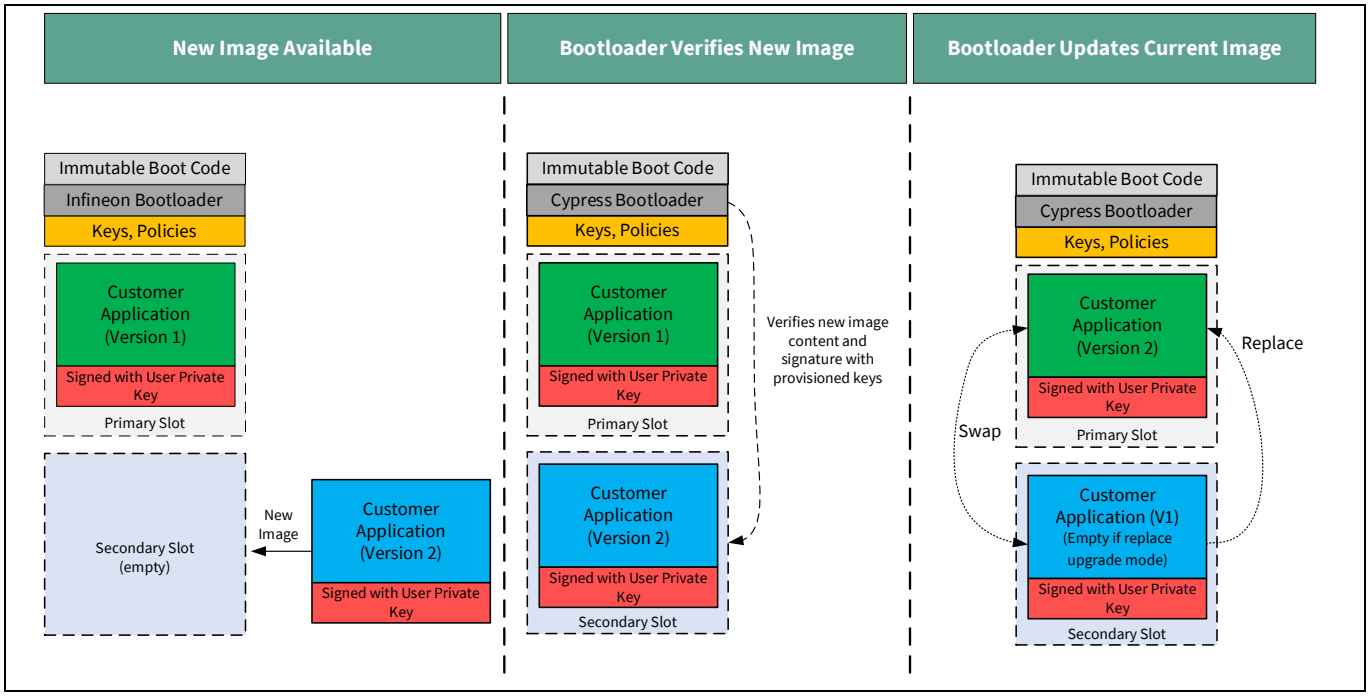

Figure 7 Bootloader application update sequence

2.5 “CySecureTools” Installation and Documentation

A stand-alone python package, “CySecureTools” contains all necessary scripts, default provisioning packets, and a set of default policy files. It implements most of the “Secure Boot” SDK functionality. “CySecureTools” is written in the Python language and requires interpreter versions higher than 3.7.

“CySecureTools” source code is available on GitHub. Full details about the operations, commands, and APIs available can be found on:

https://github.com/cypresssemiconductorco/cysecuretools/blob/master/README.md/

For Windows, the installation of ModusToolbox™ Software 2.3 or later provides all the tools required to build, program and provision devices. This includes the correct version of Python as well. Windows users may skip the remainder of this section.

Use these instructions to install and configure “CySecureTools”:

1. Install Python 3.7.4 or later on your computer. You download it from https://www.python.org/downloads/ or install it using the packet manager of host system.
2. Set up the appropriate environment variable(s) for your operating system.

   If Python 2.7 is also installed, make sure that Python37 and Python37\Scripts have higher priority in the PATH than CPython27.

   Linux

   Most distributions of Linux should already have python2 and python3 installed. To verify that python by default points to python3 run:

   ```
   python --version
   ```

   If python3 is not set as default, run the following commands. The number at the end of each command denotes a priority:

   ```
   update-alternatives --install /usr/bin/python python /usr/bin/python2.7 1
   update-alternatives --install /usr/bin/python python /usr/bin/python3.7 2
   ```

   macOS

   By default, ‘python’ points to /usr/bin/python, which is python2. To make ‘python’ and ‘pip’ resolve to python3 versions, execute the following from command line:

   ```
   echo 'alias python=python3' >> ~/.bash_profile
   echo 'alias pip=pip3' >> ~/.bash_profile
   source ~/.bash_profile
   python --version
   Python 3.7.4
   pip --version
   pip 19.0.3 from
   /Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/
   site-packages/pip (python 3.7)
   ```

   Note: If you use a shell other than bash, update its profile file accordingly. For example ~/.zshrc if you use zsh instead of ~/.bash_profile.

3. Installing “CySecureTools” package, but first make sure that you have the latest version of pip installed, use the following command.

   ```
   python -m pip install --upgrade pip
   ```

   Install the “CySecureTools” package (part of the “Secure Boot” SDK). Run the following command in your terminal window

   ```
   python -m pip install cysecuretools
   ```

   Note: During installation, you may see errors when installing colorama, protobuf and jsonschema. These can be safely ignored.

   You can use the following command to show the path to the installed package

   ```
   python -m pip show cysecuretools
   ```

4. “CySecureTools” uses the pyOCD package, which has a dependency on libusb. Follow the latest instructions in the pyOCD readme.

   The following are instructions for the currently recommended version:

   Linux

   Use the host system packet manager to install the driver using a terminal. For example, run the following for Ubuntu:

   ```
   apt-get install libusb
   ```

   This command requires sudo.

   Mac OS

   Use the homebrew packet manager to install the driver using terminal:

   ```
   homebrew install libusb
   ```

3 ModusToolbox™ Tools Provisioning Flow

This section shows how to provision the CY8CKIT-064B0S2-4343W kit in ModusToolbox™ software using "CySecureTools."

3.1 Prerequisites

3.1.1 ModusToolbox Software Installation

Install the ModusToolbox™ software, version 2.3 or later. Refer to the ModusToolbox™ installation guide.

Note: On Linux machines after installing ModusToolbox™ software, run the *ModusToolbox/tools_<version>/modus-shell/postinstall* script.

Note: ModusToolbox™ software provides a "Secure Policy" Configurator with a graphical user interface to modify policy tools and provision the device. This "Secure Boot" SDK user guide uses the command line option since it defaults to a default policy file that does not need modification for basic kit and device evaluation.

3.1.2 "CySecureTools" Installation

Follow the instructions in the "CySecureTools" Installation and Documentation section.

Note: This example will use the CY8CKIT-064B0S2-4343W kit, although feel free to use any of the PSoC™ 64 kits. The target parameter for `cysecuretools` may either be the kit name or the device family name. See table below. For the example in this guide, the device family name will be used. A user with a custom board will most likely use the device family name as well. For example, these two commands are the same:

```
cysecuretools -t cyb06xxa init
cysecuretools -t cy8ckit-064b0s2-4343w init
```

Table 2 "CySecureTools" target parameters

Kit	`cysecuretools` Target Parameter		Description
	Kit name	Device family name	
CY8CPROTO-064S1-SB	`cy8cproto-064s1-sb`	`cyb06xx7`	1M Flash
CY8CPROTO-064B0S1-BLE	`cy8cproto-064b0s1-ble`	`cyb06xx7`	1M Flash w/Bluetooth LE
CY8CKIT-064B0S2-4343W	`cy8ckit-064b0s2-4343w`	`cyb06xxa`	2M Flash
CY8CKIT-064S0S2-4343W	`cy8ckit-064s0s2-4343w`	`cyb06xxa`	2M Flash
CY8CPROTO-064B0S3	`cy8cproto-064b0s3`	`cyb06xx5`	512K Flash

3.1.3 Create Secure Blinky LED FreeRTOS Application Project

1. Launch the Eclipse IDE for ModusToolbox™.
2. Open an existing workspace or create a new workspace.
3. Click on **File > New > ModusToolbox™ IDE Application**.
4. In the Project Creator, select CY8CKIT-064B0S2-4343W kit (or whichever kit you have) and click on **Next >**.
5. Select the "Secure Blinky LED FreeRTOS" application and click **Create**. This may take a while as it pulls all required sources from respective repositories.

3.2 Device Provisioning

For evaluation, device provisioning can be done in your local development environment, rather than in a secure manufacturing facility. For evaluation, a pre-signed development token is available in the SDK which authorizes a HSM key-pair provided in the SDK.

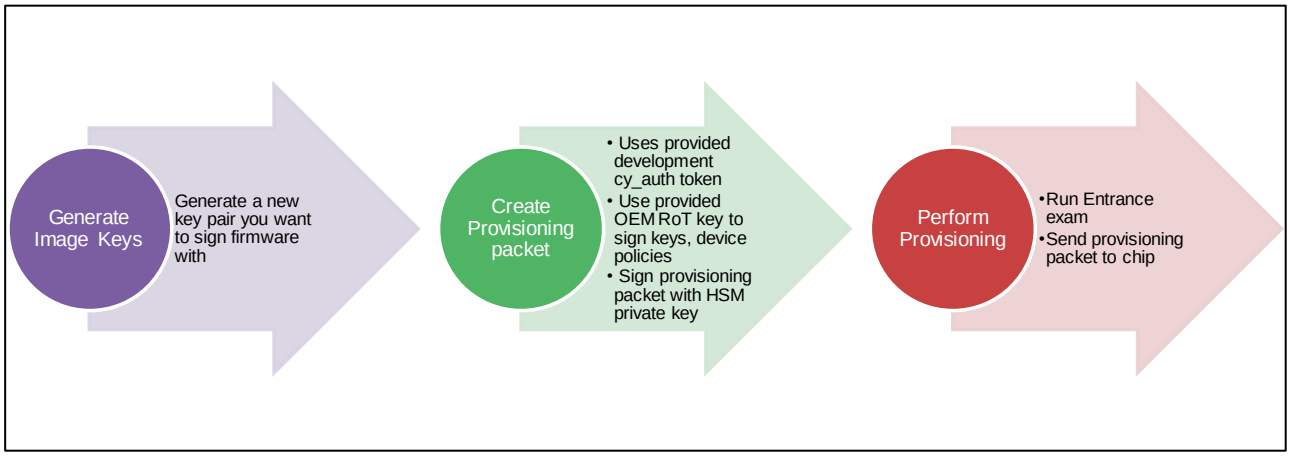

Figure 8 Provisioning flow

All the following steps should be executed from command line. The path to the policy file (if using a custom policy) can be relative to current working directory, or absolute. All paths to keys files inside policy file are absolute or relative to the policy path.

3.2.1 A. Set Up "CySecureTools" Workspace

Open a native command-line application and navigate to the *%WORKSPACE%/Secure_Blinky_LED_FreeRTOS/* directory.

For Windows, use the command line "modus-shell" program provided in the ModusToolbox™ installation instead of a standard Windows command line application. This shell provides access to all ModusTolbox™ tools including "CySecureTools" that is used to provision a device. You can access modus-shell by typing "modus-shell" in the Windows search box in the Windows menu.

The following provisioning example uses the CY8CKIT-064B0S2-4343W kit as the target. You may replace the target parameter with either the kit name or the device family name for the kit that you are using.

Run the following command: (If you have a different kit than the CY8CKIT-064B0S2-4343W, replace the target parameter "`cyb06xxa`" with the correct option for your kit for all the following commands.)

```
cysecuretools -t cyb06xxa init
```

What does this step do?

"CySecureTools" provides default policies and other secure assets that can be used to quickly set up the chip with development parameters, like leaving the CM4 DAP (Debug Access Port) open to reprogram the chip.

Based on the selected target, this step sets up all the necessary files in your workspace that are used for subsequent steps.

After running this step, you will have a choice of multiple default policies you can use to provision the chip. You can choose which policy you want to use by the `--policy/-p` flag in the "CySecureTools" CLI.

For details on what each default policy means, see Understanding the Default policy.

Note: Ensure you use the same policy file when running through steps B, C, D, and E.

Note: If you are not using the CY8CKIT-064B0S2-4343W kit, the default policy file name will be "policy_single_CM0_CM4.json". You may look in the "policy" directory to examine the example policy files.

3.2.2 B. Generate new keys

Ensure you are in the "%WORKSPACE%/Secure_Blinky_LED_FreeRTOS/" directory.

In your command-line, copy/paste the following command:

```
cysecuretools -t cyb06xxa -p policy/policy_single_CM0_CM4_swap.json create-keys
```

What does this step do?

"CySecureTools" reads the provided policy and generates the keys defined.

Depending on the policy chosen, there can be multiple keys generated under the /keys/ folder. By default only one key, the USERAPP_CM4_KEY, a P-256 Elliptic curve key-pair is generated.

"CySecureTools" generates keys in two formats, PEM and JSON. Both the PEM and JSON files represent the same key.

3.2.3 C. (Optional) Run Entrance Exam

Connect the kit to your PC.

Attention: The KitProg3 must be in DAPLink mode for performing this step. Press the 'Mode' button on the kit until the Status LED blinks fast. For more details, refer to the KitProg3 user guide.

In your command-line copy/paste the following command:

```
cysecuretools -t cyb06xxa entrance-exam
```

What does this step do?

The Entrance exam is a test routine that does the following things:

- Verify that the Device is in the correct lifecycle stage.
- Verify that Boot Code has not been modified/tampered.
- Verify that User flash is empty and no code is running before any provisioning takes place.

Failing the entrance exam returns an error in the command line. If there is any firmware running on the device, "CySecureTools" will give an option to erase the chip. Existing firmware can be erased using tools like Infineon Programmer.

Note that the entrance exam is also run automatically before performing provisioning, so you can skip this step if needed.

3.2.4 D. Perform Provisioning

Attention: KitProg3 must be in DAPLink mode. ***The kit supply voltage must be 2.5 V to perform this step***. Refer to the relevant kit user guide to find out how to change the supply voltage of your kit.

Ensure you are in the *%WORKSPACE%/Secure_Blinky_LED_FreeRTOS/* directory. In your command-line copy/paste the below command for device provisioning:

```
cysecuretools -t cyb06xxa -p policy/policy_single_CM0_CM4_swap.json
provision-device
```

What does this step do?

The "CySecureTools" provision-device API does the following steps:

- Reads the provided policy and forms the final provisioning packet, named prov_cmd.jwt.
- Performs the entrance exam.
- Provisions the device by sending the prov_cmd.jwt to the PSoC™ 64 Secure MCU.

Before running this step, you can modify the default policy to match your end use-case. For most development use-cases, you don't need to modify it. Please see Understanding the Default policy.

3.3 Device Re-provisioning

The default device policy templates provided in "CySecureTools" allows you to re-provision a device after running through the provisioning steps.

To re-provision a device, follow the steps in the normal provisioning flow (see Device Provisioning) and run the following command:

```
cysecuretools -t cyb06xxa -p policy/policy_single_CM0_CM4_swap.json re-
provision-device
```

When re-provisioning a device, the Entrance exam step is not run again. In case of failure at re-provisioning, see Re-provisioning After Failure.

3.4 ModusToolbox™ Secure Image Generation

In ModusToolbox™ software, PSoC™ 64-based kit targets have post-build signing scripts set up in the makefile so the output binary is formatted and signed automatically according to the provisioned policy file; for example, *policy_single_CM0_CM4_swap.json*.

The post-build signing is part of the .mk file located in the target; for example, *..\mtb_shared\\TARGET_CY8CKIT-064B0S2-4343w\latest-v2.X\CY8CKIT-064B0S2-4343W.mk*

The following diagram shows the flow for signing and encryption using ModusToolbox tools:

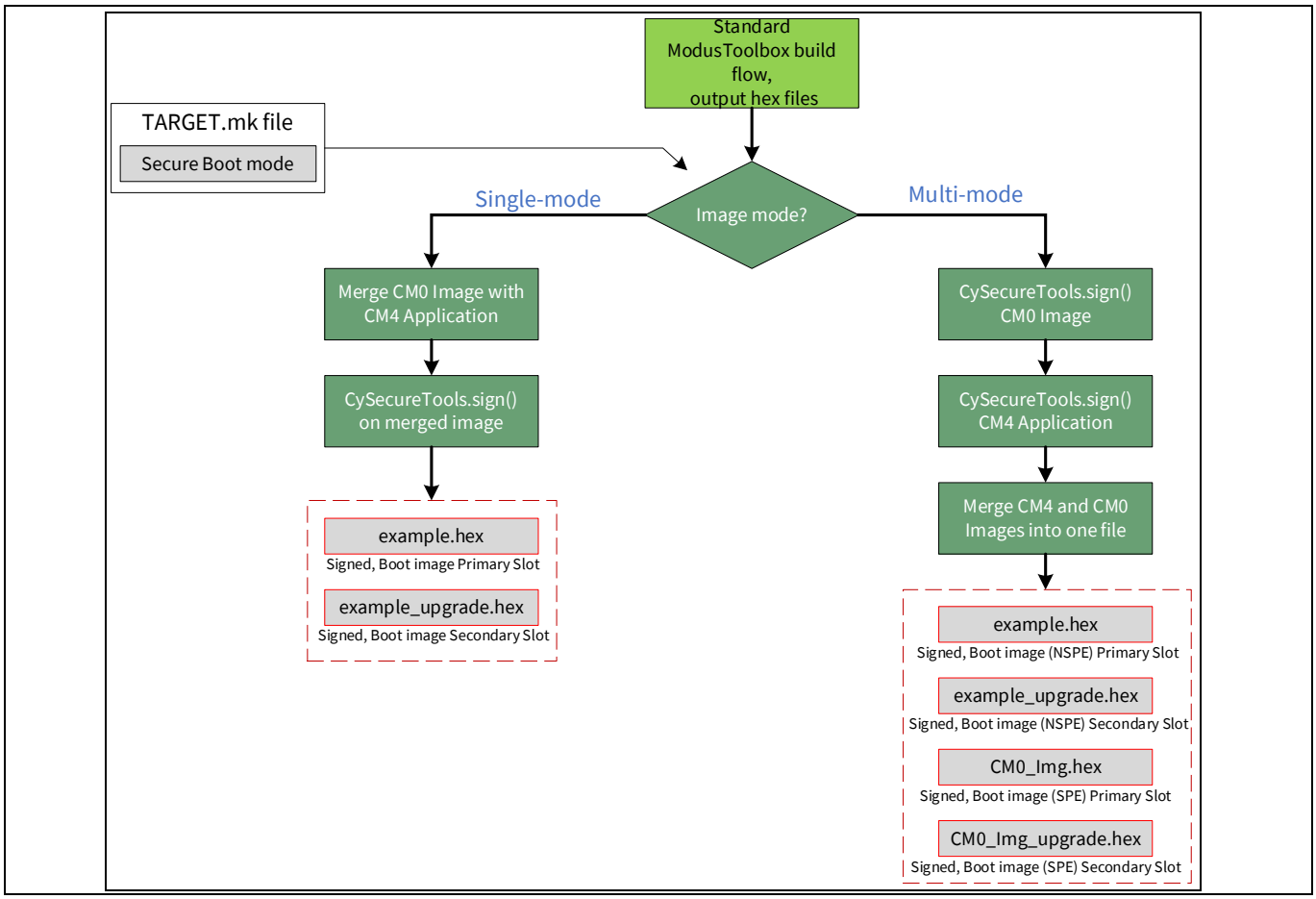

Figure 9 Secure image generation

The build process outputs two binaries:

- Signed boot image hex file. This is the exact binary that can be programmed to Primary Slot for the PSoC™ 64 device to securely launch the application.
- Signed and encrypted (policy dependent) update image hex file. This is the exact binary that can be programmed to Secondary Slot for the PSoC™ 64 device to perform a secure update and then launch the application.

3.5 Build and Run the Application

If you had just completed the provisioning process, your MiniProg4 or your kit/protoboard KitProg may still be in DAPLink mode which is required for provisioning. Be sure to change your MiniProg4 or KitProg back into CMSIS-DAP Bulk mode before attempting to program or debug your device. The MiniProg4/KitProg status LED should be on but not blinking to be in the CMSIS-DAP Bulk mode. To return the device to this mode, simply press the Mode Select button on the MiniProg4 or KitProg.

Also, you may have changed the supply voltage to 2.5 volts for provisioning. You should change the supply voltage back to your normal operating voltage prior to programming and application operation.

1. In the Project Explorer, right-click on the “Secure Blinky LED FreeRTOS” project and select **Build Project**.
2. Connect device to the computer over USB.
3. Right-click on “Secure Blinky LED FreeRTOS” project and select **Run As > Run Configurations…**
4. On the dialog, select **GDB OpenOCD Debugging > “Secure Blinky LED FreeRTOS” Program (KitProg3)** and click the **Run** button.

3.6 Debug the application

1. Right-click on “Secure Blinky LED FreeRTOS” project and select **Run As > Debug Configurations…**
2. On the dialog, select **GDB OpenOCD Debugging > “Secure Blinky LED FreeRTOS” Program (KitProg3)** and click the **Debug** button.

A breakpoint is set at main function with default launch configurations. After the first step debugger breaks at main function.

3.7 Re-provisioning after Failure

Perform this step as needed for the following scenario:

- You provision the PSoC™ 64 device, build, sign, and program the application.
- The application is verified and started by the Infineon Bootloader, but it does not work correctly and puts the device into a hard fault.
- You try to re-provision the device, using the default re-provisioning command.

  If the Primary Slot address and user keys were not changed, the Infineon Bootloader starts the bad application during re-provisioning process, the device becomes nonresponsive (the application does not produce a synchronization event for the external programming tool) and the re-provisioning process fails after timeout.

  To address this failure, erase the boot slot manually before re-provisioning, or use the following option in the re-provisioning command:

```
cysecuretools -t cyb06xxa -p policy/policy_single_CM0_CM4_swap.json re-
provision-device --erase-boot
```

Note: The re-provisioning step may fail if the device enters Deep Sleep mode. If the application code is putting the device into Deep Sleep mode, ensure to erase the flash before running the re-provisioning command.

4 "CySecureTools" Design

This section provides an overview of the "CySecureTools" python package design and details on the default policy. "CySecureTools" provides a Command-line interface over stand-alone scripts that simplifies calling them with minimum number of arguments. Advanced users can use the scripts without wrapper and configure each argument as they need. A Graphical User Interface (GUI) for "CySecureTools" is also provided. See the "Secure Policy Configurator" guide. It provides both easy execution of "CySecureTools" without using a command line and editing of the policy file without directly editing the XML policy file.

4.1 "CySecureTools" Component Diagram

The following diagram shows the high-level components of "CySecureTools":

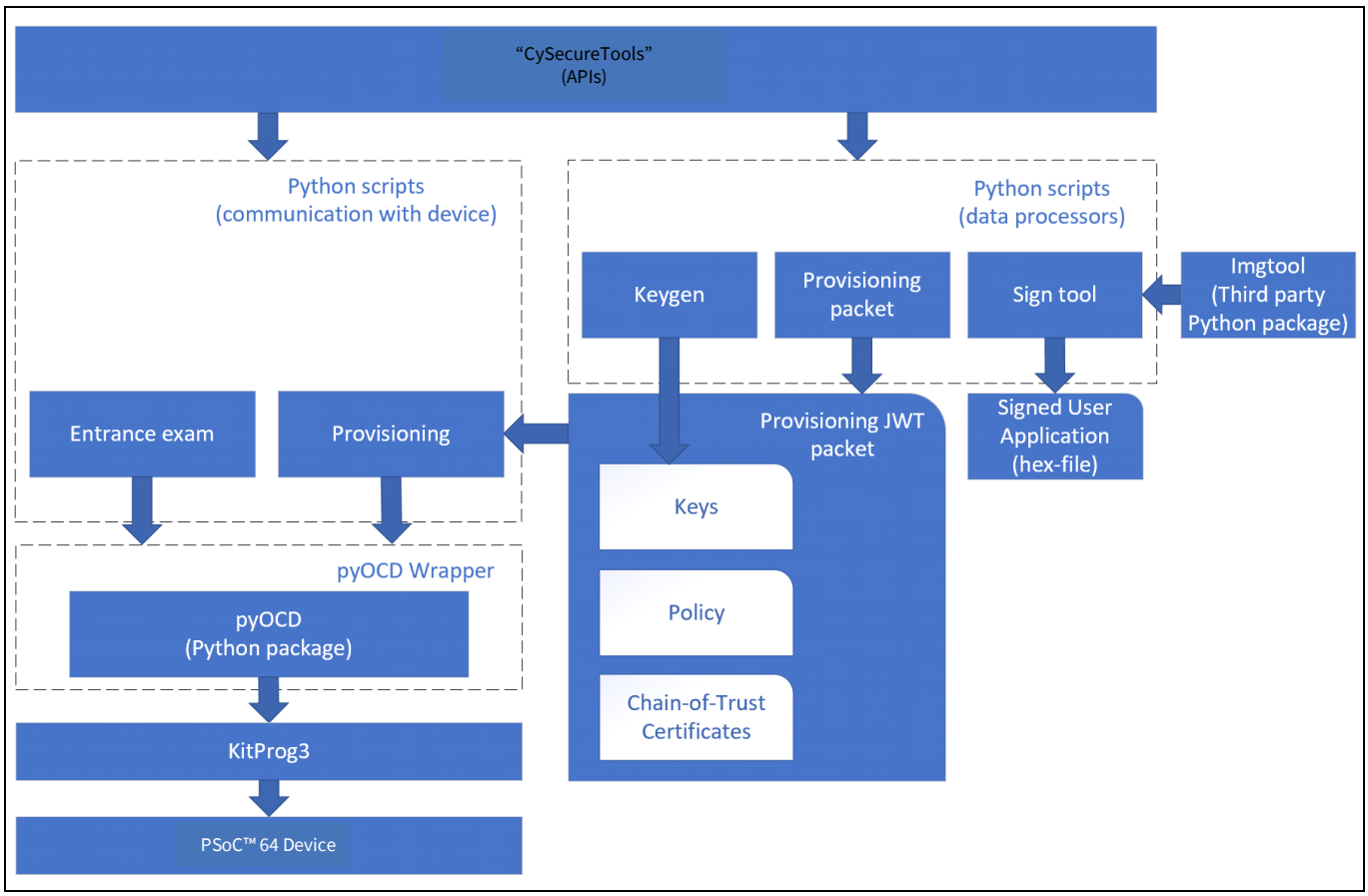

Figure 10 "CySecureTools" components

4.1.1 Creating a Provisioning Packet

The final prov_cmd.jwt which is required for provisioning a Secured MCU device requires several pieces of information.

As input arguments, the tools (specifically the create-provisioning-packet API) takes:

- OEM key file
- HSM key file
- Infineon Bootloader image certificate
- Provisioning authorization certificate

- Policy file
- Output directory (*packet* by default)
- User's keys to be used for image signing
- Chain of trust certificates

The output of the script will be added to the folder *packet*.

The *prov_identity.jwt* packet is used for giving the identity to the device during provisioning. The data provisioned with this packet cannot be changed during re-provisioning.

The *prov_cmd.jwt* file is used during provisioning and re-provisioning. This packet contains the data that can be changed during re-provisioning.

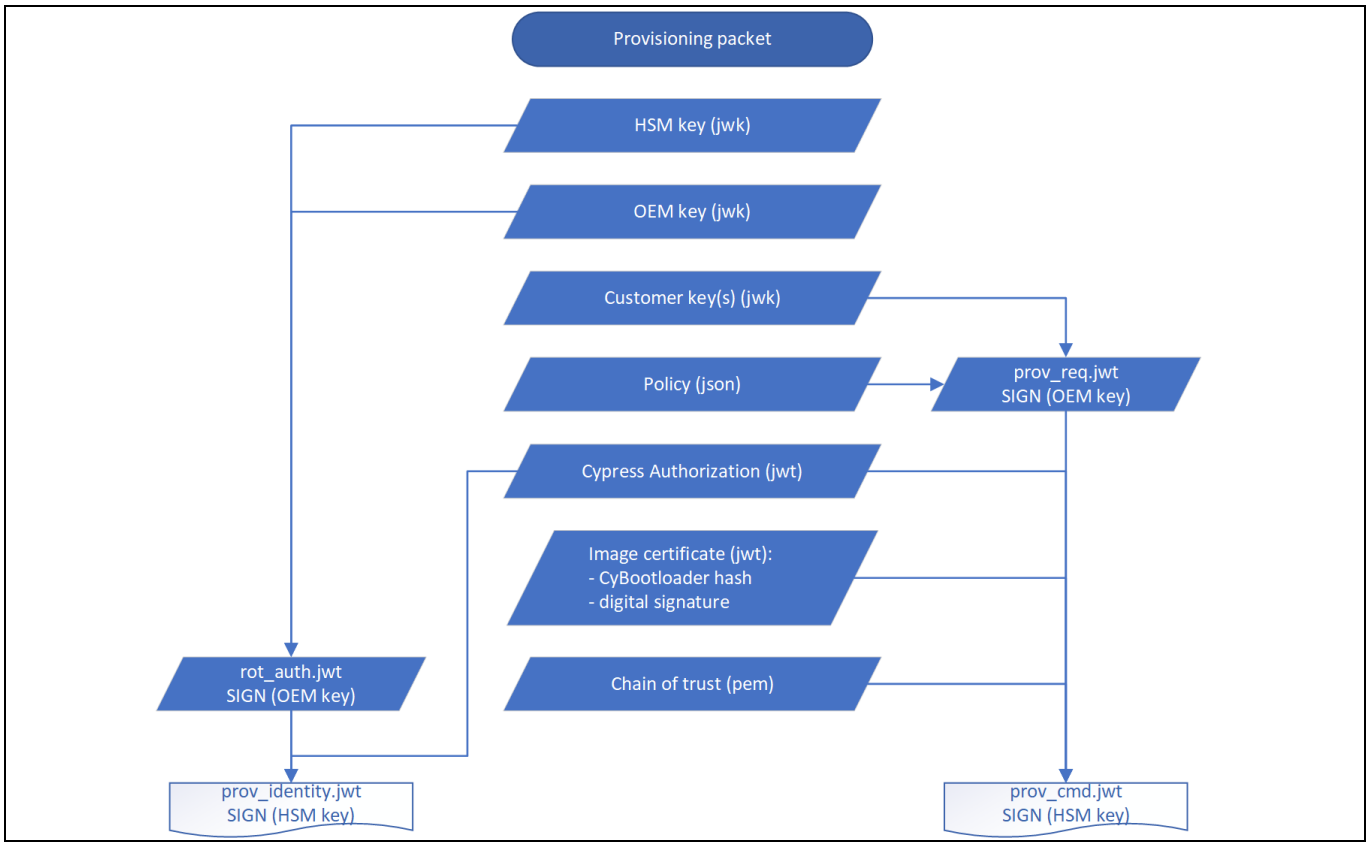

Figure 11 Provisioning packet structure

4.2 Understanding the Default Policy

There are four example policies provided in "CySecureTools" for each target. The devices with 2M or more Flash implement a swap upgrade mode instead of the standard replace mode. The policy files for these devices will contain "swap" in the name. You can see the difference in the table below for each default policy name.

Table 3 Policy files description

Policy name	Description	Debug	Application Memory Map
policy_single_CM0_CM4.json policy_single_CM0_CM4_swap.json (Default policy used in MTB)	Policy for applications which need to have single signature for 2 combined applications – Secure CM0p and User App on CM4. Typical use-case for simple secure boot and upgrade systems.	CM0P "secure" co-processor/CM4/SysAP are enabled.	Internal Flash only
policy_single_CM0_CM4_smif.json policy_single_CM0_CM4_smif_swap.json	Similar to previous policy, Enables external memory support for upgrade image location	CM0P "secure" co-processor/CM4/SysAP are enabled.	External memory support for upgrade image
policy_multi_CM0_CM4.json policy_multi_CM0_CM4_swap.json	Policy for applications which need independently updateable SPE and NSPE code. Typical use-case for IoT systems which maintain Secured code independent of application code.	CM0P "secure" co-processor/CM4/SysAP are enabled.	Internal Flash only
policy_multi_CM0_CM4_smif.json policy_multi_CM0_CM4_smif_swap.json	Similar to previous policy, Enables external memory support for upgrade image location.	CM0P "secure" co-processor/CM4/SysAP are enabled	External memory support for upgrade images.

This section covers the details of the fields in the default policy_single_CM0_CM4/_swap policy provided in the "CySecureTools." The contents can be classified as follows:

- Boot&Upgrade Policy
- Debug Policy
- Infineon Bootloader
- "CySecureTools" miscellaneous assets

4.2.1 Policy and Configuration Limitations

The follow is a list of limitations with the policy and configuration of the PSoC™ 64 "Secure Boot MCU."

- Only upgrade (Secondary Slot) may reside in external memory (SMIF-based). The boot (Primary Slot) must reside in internal flash.
- Only one SMIF slave-select may be used in a system with firmware upgrades enabled. Other SMIF devices with additional slave-selects may be used for non-code related storage.
- Secondary Slot(s) in external memory start address must start at 0x1800_000 + (X * SMIF sector_size), where X = 0, 1, etc. The external memory devices on the PSoC™ 64 kits have a sector size of 0x40000 (256K). For example, the external memory starts at address 0x1800_0000 and the sector size is 0x40000, then the start address may be 0x1800_0000, 0x1804_0000, 0x1808_0000, etc.
- Only SFDP (Serial Flash Discoverable Parameter) compatible devices are supported for external memory.
- Larger sector size must be an even multiple number of the smaller sector size. In the case of the current PSoC™ 64 development kits, the external memory sector size is 0x40000 (larger) and the internal sector size is 0x200 (smaller), which is an even multiple of the larger.
- The same key pair must be used to sign both the boot and the upgrade images.
- The internal slot size (Primary and Secondary) for SWAP must be equal or larger of the sum of the image size (even multiple of sector size), one move sector (512 bytes) and image trailer size (512 bytes).
- The Boot/Upgrade Image, Move_Sector, and Trailer must be on non-overlapping sectors. This means that external memory with a sector size of 0x40000 (256K) will need to be at least 3 times the size of the sector size. In this case it would be 0xC0000 (768K), since the Move_Sector and Trailer each have to be at least 256K (1 sector).
- Primary and Secondary Slot sizes must be equal.
- External clocks (ECO, ALTHF, WCO, and EXTCLK) are not allowed to source CLK_HF0 if they are not defined in the security policy.
- Devices with 2M or more of flash will only use SWAP mode for firmware upgrades and devices with less than 2M Flash will used REPLACE upgrade mode.

4.2.2 Boot&Upgrade Policy

The Boot&Upgrade (BnU) policy defines the memory regions and keys associated with images in the chip. This JSON field has further sub-objects:

- Infineon BnU Policy
- CM0+ "Secure" co-processor Image BnU Policy
- CM4 Image BnU Policy
- Reprovisioning options

The tables on the next few pages show the Infineon Bootloader settings in the *example/default policy_single_CM0_CM4_swap.json* file for the cyb06xxa family (2M Flash) devices. This example policy file is part of the "CySecureTools" version 3.1. These fields may need to be modified, but they can be useful as a reference.

The following figure shows the memory map of the flash defined by the policy file. See Appendix A for memory map files for the example policy files for "CySecureTools" 3.1.

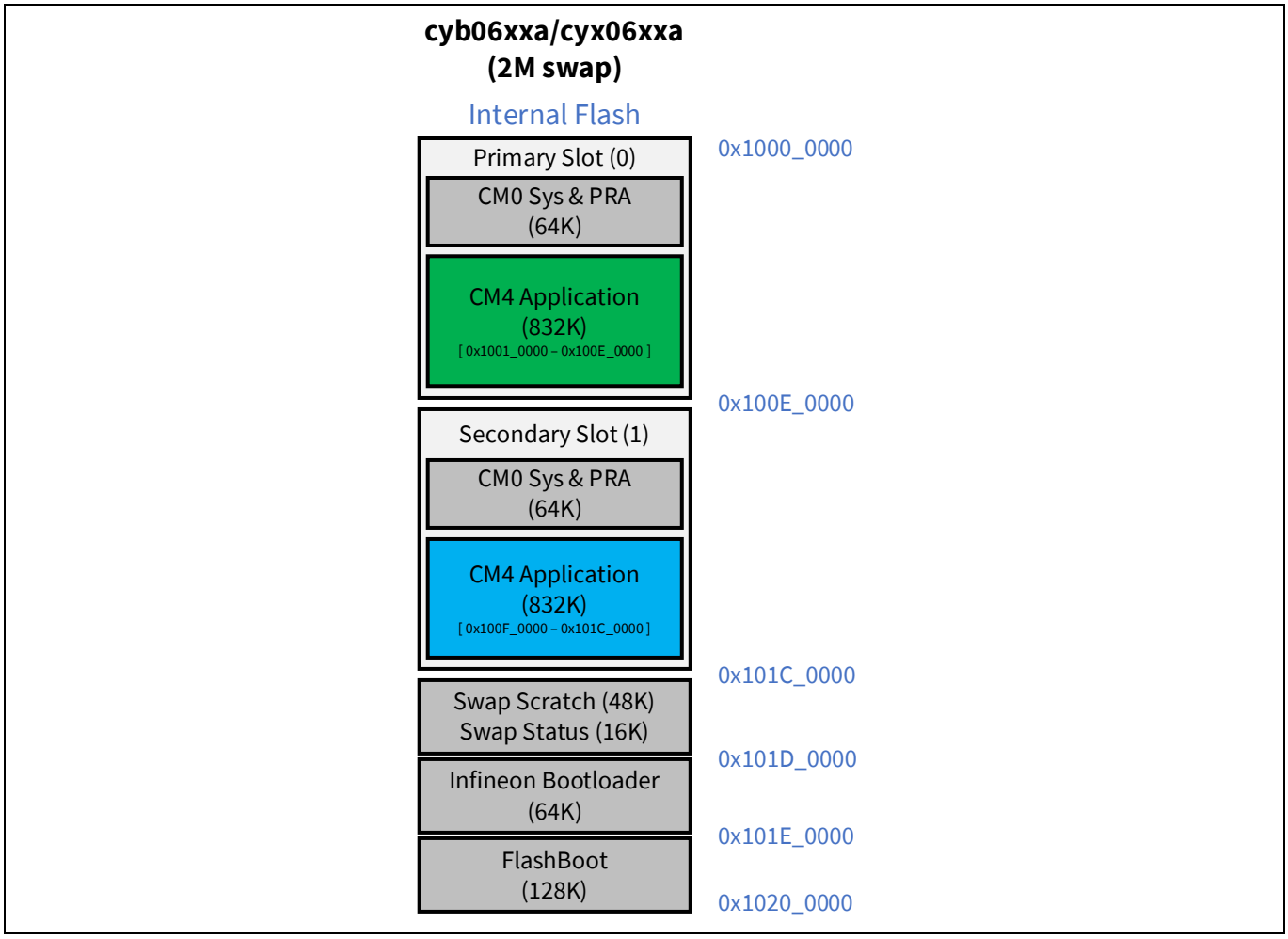

Figure 12 **PSoC™ 64 "Secure" MCU flash memory map**

Table 4 **Infineon Bootloader Boot&Upgrade policy**

JSON field	Description
`{`	
`"boot_auth": [5],`	KeyID 5 used check image signature
`"bootloader_keys": [`	Defines key used for Bootloader
`{`	
`"kid": 5,`	Specify KeyID = 5 for the key
`"key": "../keys/oem_state.json"`	Path to key
`}`	
`],`	
`"id": 0,`	Image Id '0', Indicates Infineon Bootloader
`"launch": 1,`	Next image to launch is identifier '1', indicates CM0+ "secure" co-processor image

JSON field	Description
`"upgrade_mode": "swap",`	This device is using "swap" mode for firmware upgrade. The 512K and 1M flash devices use "replace" mode which is the default if this policy is not in the policy.
`"acq_win": 100,`	Defines acquire window for Infineon Bootloader in msec.
`"wdt_timeout": 4000,`	WDT for the CM0+ "secure" co-processor set to 4000 msec.
`"wdt_enable": true`	WDT for CM0+ "secure" co-processor is enabled.
`"monotonic": 0,`	The counter to protect the rollback during the upgrade process.
`"clock_flags": 578,`	Clock flag - 0x0242; Listen window is 100 ms; CM0+ "secure" co-processor clock set to 50 MHz when executing Infineon Bootloader.
`"protect_flags": 1,`	Private key protection Enabled Bit0 – Private key protection (0 - Disabled, 1 - Enabled) Bit1 – Any PSA API protection (0 - Disabled, 1 - Enabled)
`"upgrade": false,`	Bootloader is not upgradable.
`"resources": [`	Defines Resources used by image.
`{`	
`"type": "FLASH_PC1_SPM",`	Indicates Flash region to be protected at PC = 1.
`"address": 270336000,`	Address: 0x101D_0000
`"size": 65536`	Size: 64K
`},`	
`{`	
`"type": "SRAM_SPM_PRIV",`	Indicates RAM region to be protected at PC = 1.
`"address": 135135232,`	Address: 0x080E_0000
`"size": 65536`	Size: 64K
`},`	
`{`	
`"type": "SRAM_DAP",`	Indicates RAM reserved by DAP for debugging.
`"address": 135184384 ,`	Address: 0x080E_C000
`"size": 16384`	Size: 16K
`}`	
`{`	
`"type": "STATUS_PARTITION"`	Status partition used for SWAP firmware updates.
`"address": 270319616`	0x0x101C_C000
`"size" 16384`	0x4000 (16K)
`}`	

JSON field	Description
{	
"type": "SCRATCH"	Scratch flash memory used for SWAP firmware updates.
"address": 270270464,	0x101C_0000
"size": 49152	0xC000 (48K)
]	
},	

Table 5 CM0+ "Secure" Co-processor Application image Boot&Upgrade policy

JSON field	Description
{	
"boot_auth": [8],	KeyID 8 used to check image signature.
"boot_keys": [	Defines key used by Infineon Bootloader.
{	
"kid": 8,	Specify KID = 8 for the below key.
"key": "../keys/USERAPP_CM4_KEY.json"	Path to Key
}	
],	
"id": 1,	Image Id '1', Indicates CM0+ "secure" co-processor Image
"launch": 16,	Image id '16' (CM4 App) will be launched by this image.
"monotonic": 0,	The counter to protect the rollback during the upgrade process.
"smif_id": 0,	No upgrade image in external memory.
"acq_win": 100,	Defines acquire window for this image in msec.
"wdt_timeout": 4000	Set CM0+ "secure" co-processor watchdog timer to 4 seconds.
"wdt_enable": false	Do not enable watchdog timer.
"set_img_ok": true	Set to one to set current image valid.
"upgrade": true,	This image is upgradable from secondary slot (1).
"version": "0.1",	Version of image, used by "CySecureTools" to form MCUBoot header.
"rollback_counter:" 0,	One-way version counter of zero.
"encrypt": false,	Encryption for upgrade slot is disabled.
"encrypt_key_id": 1,	Kid: 1(device private key) used for ECDH for deriving key of encrypted update.

JSON field	Description
"encrypt_peer": "../keys/dev_pub_key.pem",	Path to public key to be used by "CySecureTools" to for the key.
"resources": [	Defines Resources used by image.
{	
"type": "BOOT",	Indicates Primary Slot (0)
"address": 268435456,	Address: 0x1000_0000
"size": 917504	Size: 896K, 0xE_0000
},	
{	
"type": "UPGRADE",	Indicates Secondary Slot(1).
"address": 269459456,	Address: 0x100E_0000
"size": 917504	Size: 896K, 0xE_0000
}	
]	
},	

Special Note: In the single-image use case, most fields in the M4 Boot&Upgrade image policy are placeholders. All the required information (except CM4 Boot address) is derived from the CM0+ "Secure" co-processor policy which has the combined firmware image for both cores.

Table 6 CM4 Application image Boot&Upgrade policy

JSON field	Description
{	
"boot_auth": [8],	N/A (Only valid with dual image)
"boot_keys": [	N/A (Only valid with dual image)
{	
"kid": 8,	N/A (Only valid with dual image)
"key": "../keys/USERAPP_CM4_KEY.json"	N/A (Only valid with dual image)
}	
],	
"id": 16,	Image Id '16', Indicates CM4 Image. Since this is a single image the eight values below are ignored. See id: 1 above for settings.
"monotonic": 8,	N/A (Not valid with single image)
"smif_id": 0,	N/A (Not valid with single image)
"upgrade": false,	N/A (Not valid with single image)
"version": "0.1",	N/A (Not valid with single image)
rollback_counter: 0,	N/A (Not valid with single image)
"encrypt": false,	N/A (Not valid with single image)

JSON field	Description
"encrypt_key_id": 1,	N/A (Not valid with single image)
"encrypt_peer": "../keys/dev_pub_key.pem",	N/A (Not valid with single image)
"resources": [	Defines Resources used by image
{	
"type": "BOOT",	Indicates Launch address of CM4 part of single image.
"address": 268500992,	Address: 0x10010000
"size": 884736	N/A (Not valid with single image)
}	
],	

Table 7 Reprovisioning options

JSON field	Description
{	
"boot_loader": true,	Bootloader can be re-provisioned.
"keys_and_policies": true	Keys and Policies can be re-provisioned.
}	

4.2.3 Debug Policy

The Debug policy specifies how various access ports are configured for the part.

Table 8 Infineon Debug policy

JSON field	Description
{	
"m0p": {	Defines CM0P "secure" co-processor DAP Port behavior.
"permission": "enabled",	DAP Port enabled
"control": "firmware",	N/A (Only valid if permission set to Allowed)
"key": 5	N/A (Only valid if permission set to Allowed)
},	
"m4": {	Defines CM4 DAP Port behavior.
"permission": "allowed",	DAP Port enabled
"control": "firmware",	N/A (Only valid if permission set to Allowed)
"key": 5	N/A (Only valid if permission set to Allowed)
},	
"system": {	Defines CM4 DAP Port behavior.
"permission": "enabled",	DAP Port enabled
"control": "firmware",	N/A (Only valid if permission set to Allowed)

JSON field	Description
`"key": 5,`	N/A (Only valid if permission set to Allowed)
`"flashw": true,`	Allow Flash Writes using SysAP port.
`"flashr": true`	Allow Flash Reads using SysAP port.
`},`	
`"rma": {`	Defines RMA behavior.
`"permission": "allowed",`	RMA mode is allowed.
`"destroy_fuses": [`	Indicates eFuse region to be destroyed if entering RMA mode.
`{`	
`"start": 888,`	Start address of efuses to be destroyed.
`"size": 136`	Size in bits to be destroyed.
`}`	
`],`	
`"destroy_flash": [`	Indicates Flash region to be destroyed if entering RMA mode.
`{`	
`"start": 268435456,`	Start Address of flash to be destroyed (0x10000000).
`"size": 512`	Size in bytes of flash to be destroyed.
`}`	
`],`	
`"key": 5`	KeyID used to validate a RMA request.
`}`	
`}`	

4.2.4 External Clock Policy

The External Clock policy specifies the port/pins and frequency of an external clock.

Table 9 External Clock policy

JSON field	Description
`{`	
`"custom_data_sections": ["extclk", "srampwrmode"],`	
`"extclk": {`	Defines external clock options.
`"extClkEnable": 0,`	Enable extClk = 1, Disable extClk = 0
`"extClkFreqHz": 4000000,`	extClk clock frequency = 4.00 MHz
`"extClkPort": 0,`	extClk port 0
`"extClkPinNum": 5,`	extClk pin 5 P0[5]
`"ecoEnable": 0,`	Enable ECO = 1, Disable ECO = 0
`"ecoFreqHz": 24000000,`	ECO frequency = 24.00 MHz

JSON field	Description
`"ecoLoad": 20,`	ECO Load 20 pF (See TRM)
`"ecoEsr": 30,`	ECO ESR 30 (See TRM)
`"ecoDriveLevel": 100,`	ECO Drive Level 100 (See TRM)
`"ecoInPort": 12,`	ECO input port 12
`"ecoOutPort": 12,`	ECO output port 12
`"ecoInPinNum": 6,`	ECO input pin P12[6]
`"ecoOutPinNum": 7,`	ECO output pin P12[7]
`"bypassEnable": 0,`	Clock port bypass to External sine wave or crystal (See TRM).
`"wcoEnable": 1,`	Enable WCO = 1, Disable WCO = 0
`"wcoInPort": 0,`	WCO input port 0
`"wcoOutPort": 0,`	WCO output port 0
`"wcoInPinNum": 0,`	WCO input pin P0[0]
`"wcoOutPinNum": 1`	WCO output pin P0[1]
`}`	

4.2.5 Infineon Bootloader

```
"cy_bootloader":
{
      "mode" : "debug", -> CySecureBootloader will emit debug logs over UART
}
```

4.2.6 "CySecureTools" Misc Assets

Table 10 "CySecureTools" Misc Assets

JSON field	Description
`"provisioning": {`	Defines provisioning packet paths.
`"packet_dir": "../packets",`	Relative path of the packets folder used for provisioning.
`"chain_of_trust": []`	No chain-of-trust certificate objects.
`},`	
`"pre_build": {`	Defines pre-build asset locations needed for provisioning.
`"oem_public_key": "../keys/oem_state.json",`	Relative path for OEM root public key location.
`"oem_private_key": "../keys/oem_state.json",`	Relative path for OEM root private key location.

JSON field	Description
"hsm_public_key": "../keys/hsm_state.json",	Relative path for HSM public key location.
"hsm_private_key": "../keys/hsm_state.json",	Relative path for HSM private key location.
"provision_group_private_key": false,	No group private keys provisioned.
"group_private_key": "../keys/grp_priv_key.json",	Relative path for group private key location.
"provision_device_private_key": false,	No device private key provisioned.
"device_private_key": "../keys/dev_priv_key.json",	Relative path for device private key location.
"cy_auth": "../packets/cy_auth_1m_b0_sample.jwt"	Relative path for cy_auth location.
}	

4.3 Provisioning JWT packet Reference

4.3.1 prov_cmd.jwt

The prov_cmd.jwt is the final packet sent to the PSoC™ 64 "Secure Boot MCU" to finalize provisioning. The following shows this JWT structure:

Structure:

```
{
  {
  "cy_auth": "........",
  "rot_auth": "........",
  "image_cert": "........",
  "prov_req": "........",
  "chain_of_trust": [],
  "complete": Boolean Value,
  "type": "HSM_PROV_CMD"
} sig: HSM_PRIV_KEY
```

Table 11 prov_cmd.jwt parameters description

Object	Description
cy_auth	Infineon Authorization JWT, authorizes the HSM public key.
rot_auth	OEM/User authorization JWT, authorizes the HSM public key.

Object	Description
image_cert	Infineon Bootloader image JWT, used for sending a Infineon Bootloader signature.
chain_of_trust	Holds an array of X.509 certificates.
complete	True - indicates if complete provisioning process must be complete and move chip life-cycle.
type	Specifies the JWT type as a string.

4.3.2 prov_identity.jwt

The prov_identity.jwt is the initial token which is sent to the chip to create a unique identity.

Structure:

```
{
  {
  "create_identity": Boolean Value,
  "cy_auth": "……",
  "rot_auth": "……",
  "type": "HSM_PROV_CMD"
} sig: HSM_PRIV_KEY
```

Table 12 prov_identity.jwt parameters description

Object	Description
create_identity	If true, chip will form a unique identity and export the public key.
cy_auth	Infineon Authorization JWT, authorizes the HSM public key.
rot_auth	OEM/User authorization JWT, authorizes the HSM public key.
type	Specifies the JWT type as a string.

4.3.3 cy_auth.jwt

Structure:

```
{
  "auth": {},
  "cy_pub_key": {Cypress root pub key},
  "hsm_pub_key": {HSM pub key},
  "exp": {Expiry time},
  "type": "CY_AUTH_HSM"
```

```
} sig: CYPRESS_ROOT_PRIV_KEY
```

Table 13 cy_auth.jwt parameters description

Object	Description
auth	Can specify authorization limits based on device die_id.
cy_pub_key	Infineon Root Public key in the JWK format.
hsm_pub_key	HSM Root Public key in the JWK format.
exp	Specifies when the token expires in UNIX time.
type	Specifies the JWT type as a string.

4.3.4 rot_auth.jwt

Structure:

```
{
  "hsm_pub_key": {HSM pub key},
  "oem_pub_key": {OEM RoT pub key},
  "iat": {Issue time},
  "prod_id": {Product Name},
  "type": "OEM_ROT_AUTH"
} sig: OEM_RoT_PRIV_KEY
```

Table 14 rot_auth.jwt parameters description

Object	Description
hsm_pub_key	HSM Root Public key in the JWK format.
oem_pub_key	OEM RoT Public key in the JWK format.
iat	Specifies when the token was issued.
prod_id	The product string, specified by the user. Note that this MUST match prod_id in the prov_req.JWT.
type	Specifies the JWT type as a string.

4.3.5 prov_req.jwt

Structure:

```
{
  "custom_pub_key": [{Key1}, …],
```

```
  "boot_upgrade": {…},
  "debug": {…}
  "prod_id": "my_thing",
  "wounding": {}
} sig: OEM_RoT_PRIV_KEY
```

Table 15 prov_req.jwt parameters description

Object	Description
custom_pub_key	The array of customer public keys to be injected in the JWK format.
boot_upgrade	Boot and Upgrade Policy JSON
debug	Debug policy JSON
prod_id	The product string, specified by the user. Note that this MUST match prod_id in the rot_auth.JWT.
wounding	Reserved

4.3.6 boot_upgrade.JSON

Structure:

```
{
    "title": "upgrade_policy"
    "firmware": [
      {
        "boot_auth": [Integer Value],
        "bootloader_keys": [
            "kid": [Integer Value],
            "key": [String Path to key],
        "id": [Integer Value],
        "launch": Integer Value,
        "monotonic": [Integer Value],
        "resources": [
          {
            "address": Integer Value,
            "size": Integer Value,
            "type": [STRING VALUE]
          },
        ],
        "smif_id": Integer Value,
```

```
            "upgrade": Boolean Value,
            "upgrade_auth": [Integer Value]
        }, …
    ],
}
```

Table 16 boot_upgrade.JSON parameters description

Object	Description	Range of valid values
id	Image id. (0-16: Cypress reserved, >16: customer specific)	A range of integers can be specified, "0": The first firmware image started from RomBoot/FlashBoot (i.e. the boot loader). "1": CM0+ "secure" co-processor Image "2": CM0+ "secure" co-processor Infineon trusted functions "3": OEM trusted functions "4": CM4 Boot Image (direct from flashboot, not used) "16": CM4 Image
boot_auth	Specifies key index to use for validating the signature. These signatures are all verified during boot.	Can be any integer public key >3. For Infineon Bootloader, the auth is "3". For the M4 image, this can be any number depending on key_id specified in the JWK format in the custom_pub_key fields.
launch	Specifies next image 'id' being launched	"4" is the only valid value for Infineon Bootloader and the Single image bootloader case.
monotonic	Indicates the monotonic counter number associated with this image. During secure boot this counter value is compared with the current_version code in the image being booted. During upgrade this counter is incremented to the value from the image header of the upgrade image.	0~15. Counters can be rolled up by the system firmware using SysCalls.
resources: address	Specifies the start address of the image.	The valid flash range address. Only decimal values are allowed, e.g.: 268435456 -> 0x10000000
resources: size	Specifies the size of the image.	The valid flash range size in bytes. Only decimal values are allowed, e.g.: 327680-> 0x50000 -> 320 KB

Object	Description	Range of valid values
`resources: type`	Specifies type of image.	Only "BOOT" and "UPGRADE" are user-modifiable fields for the M4 image. "BOOT" -> Slot#0 "UPGRADE" -> Slot#1
`smif_id`	Specifies if external memory is used for placing Slot#1 image.	"0" – SMIF is disabled. "1" – If the CY8CPROTO_064_SB target is used.
`upgrade`	Specifies if updating is allowed for this image id.	'true' -> Upgrades are allowed. 'false' -> Upgrade is not allowed.
`upgrade_auth`	Specifies key index to use for validating the signature of the upgrade. Allows upgrades to be checked by a different key if necessary.	Can be any integer public key >3. For Infineon Bootloader, the auth is "3". For the M4 image, this can be any number depending on key_id specified in the JWK format in the custom_pub_key fields.

4.3.7 debug.JSON

Structure:

```
{
    "m0p" : {
        "permission" : " STRING VALUE ",
        "control" : " STRING VALUE ",
        "key" : [Integer Value]
    },
    "m4" : {
        "permission" : " STRING VALUE ",
        "control" : " STRING VALUE ",
        "key" : [Integer Value]
    },
    "system" : {
        "permission" : " STRING VALUE ",,
        "control" : " STRING VALUE ",,
        "key" : [Integer Value],
        "flashw": Boolean Value,
        "flashr": Boolean Value,
    },
    "rma" : {
        "permission" : "STRING VALUE ",
        "destroy_fuses" : [
```

```
            {
                "start" : Integer Value,
                "size" : Integer Value
            }
        ],
        "destroy_flash" : [
            {
                "start" : Integer Value,
                "size" : Integer Value
            },
        ],
        "key" : Integer Value
    }
}
```

Table 17 debug.JSON parameters description

Object	Description	Range of valid values
`m0p/m4/system: permission`	Specifies the permission level for the associated DAP port.	“Enabled” – The DAP port is open after bootup. “Allowed” – The DAP port can be opened after bootup, see the “control” field. “Disabled” – The DAP port is closed after bootup.
`m0p/m4/system: control`	Specifies how the DAP port can be opened after bootup. The field is only valid if “permission” is “Allowed”.	“firmware” – The code the user can choose to open the DAP port depending on some custom code. “certificate” – A signed token must be presented using a SysCall to open the DAP port.
`m0p/m4/system: key`	Specifies which Key Id to use for certificate validation in “control” field.	The key ID must be >3, point to the key provisioned in the custom_pub_key field.
`system: flashr/flashw`	Specifies which regions the SysAP port is allowed to access.	“true” -> Flash reads/writes via SysAP allowed. “false” -> Flash reads/writes via SysAP not allowed.
`rma: permission`	Specifies if RMA is allowed.	“Disabled” – RMA is not allowed. “Allowed” – The RMA stage is available and can be entered by presenting a certificate using key> to a SysCall API. The system will destroy fuse and flash contents as specified in <destroy_fuses> and <destroy_flash> before transitioning to RMA stage.

Object	Description	Range of valid values
`rma: destroy_fuses: start`	Starting fuse bit number for region.	0~65536. Check the part datasheet for the eFuse allowed address.
`rma: destroy_fuses: size`	Number of fuse bits in region.	0~65536. Check the part datasheet for the eFuse allowed size.
`rma: destroy_flash: start`	Starting byte address of region (will be rounded down to nearest program/erase boundary).	0~0xFFFFFFFF. Check the part datasheet for the flash allowed address.
`rma: destroy_flash: size`	Size in bytes of region (will be rounded up so region is integral number of program/erase units).	0~0xFFFFFFFF. Check the part datasheet for the flash allowed size.
`rma: key`	The key slot number of the key used to validate authorization to enter RMA stage.	The key ID must be >3, point to the key provisioned in the custom_pub_key field.

5 Additional resources

This chapter contains various links to additional resources that can be useful when working with the “Secure Boot” SDK.

5.1 Application notes

Table 18 Application notes

Number/Link	Title	Description
AN228571	Getting started with PSoC™ 6 MCU on ModusToolbox™	Describes PSoC™ 6 MCU devices and how to build your first application with ModusToolbox™.
AN210781	Getting started with PSoC™ 6 MCU with Bluetooth Low Energy connectivity on PSoC™ Creator	Describes PSoC™ 6 MCU with Bluetooth LE connectivity devices and how to build your first application with PSoC™ Creator.

5.2 Code example

- Using ModusToolbox™

5.3 Device documentation

- PSoC™ 6 MCU datasheets
- PSoC™ 6 Technical reference manuals

5.4 Development kits

- CY8CKIT-064B0S2-4343W PSoC™ 64 “Secure Boot” Wi-Fi pioneer kit
- CY8CPROTO-064S1-SB PSoC™ 64 “Secure Boot” prototyping kit
- CY8CPROTO-064B0S1-BLE PSoC™ 64 Bluetooth LE “Secure Boot” prototyping kit
- CY8CPROTO-064B0S3 PSoC™ 64 “Secure Boot” prototyping kit

5.5 Libraries (on GitHub)

Table 19 Libraries (on GitHub)

Name/Link	Description	Documentation
mtb-pdl-cat1	PSoC™ 6 peripheral driver library (PDL)	API Reference
mtb-hal-cat1	Hardware Abstraction Layer (HAL) Library	API Reference
retarget-io	Retarget-IO - A utility library to retarget the standard input/output (STDIO) messages to a UART port	API Reference
p64_utils	PSoC™ 64 “Secure Boot” Utilities Middleware Library	API Reference

5.6 PSoC™ 6 Middleware (on GitHub)

The following link opens a GitHub page to the section for PSoC™ 6 middleware:

- https://github.com/Infineon/modustoolbox-software#psoc-6-middleware-libraries

5.7 Tools

The following link opens the ModusToolbox™ Software and Tools webpage:

- https://www.infineon.com/modustoolbox-software

The following link opens the “CySecureTools” page on pypi.org"

- https://pypi.org/project/cysecuretools/

6 Appendix A: Flash Memory Maps

6.1 Flash memory map for policy_single_CM0_CM4 policy files

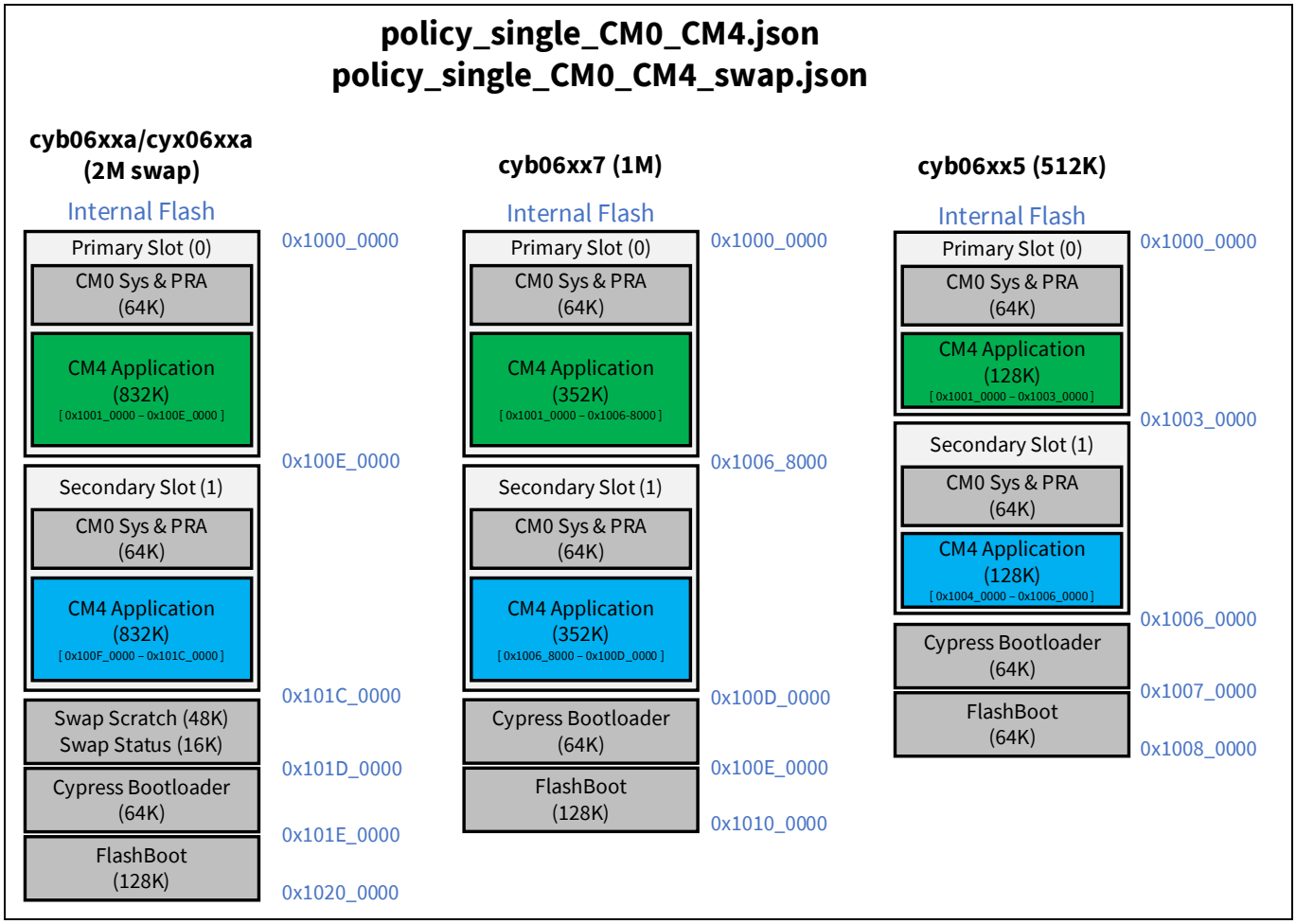

Figure 13 **Flash memory map for policy_single_CM0_CM4 policy files**

6.2 Flash memory map for policy_multi_CM0_CM4 example policies

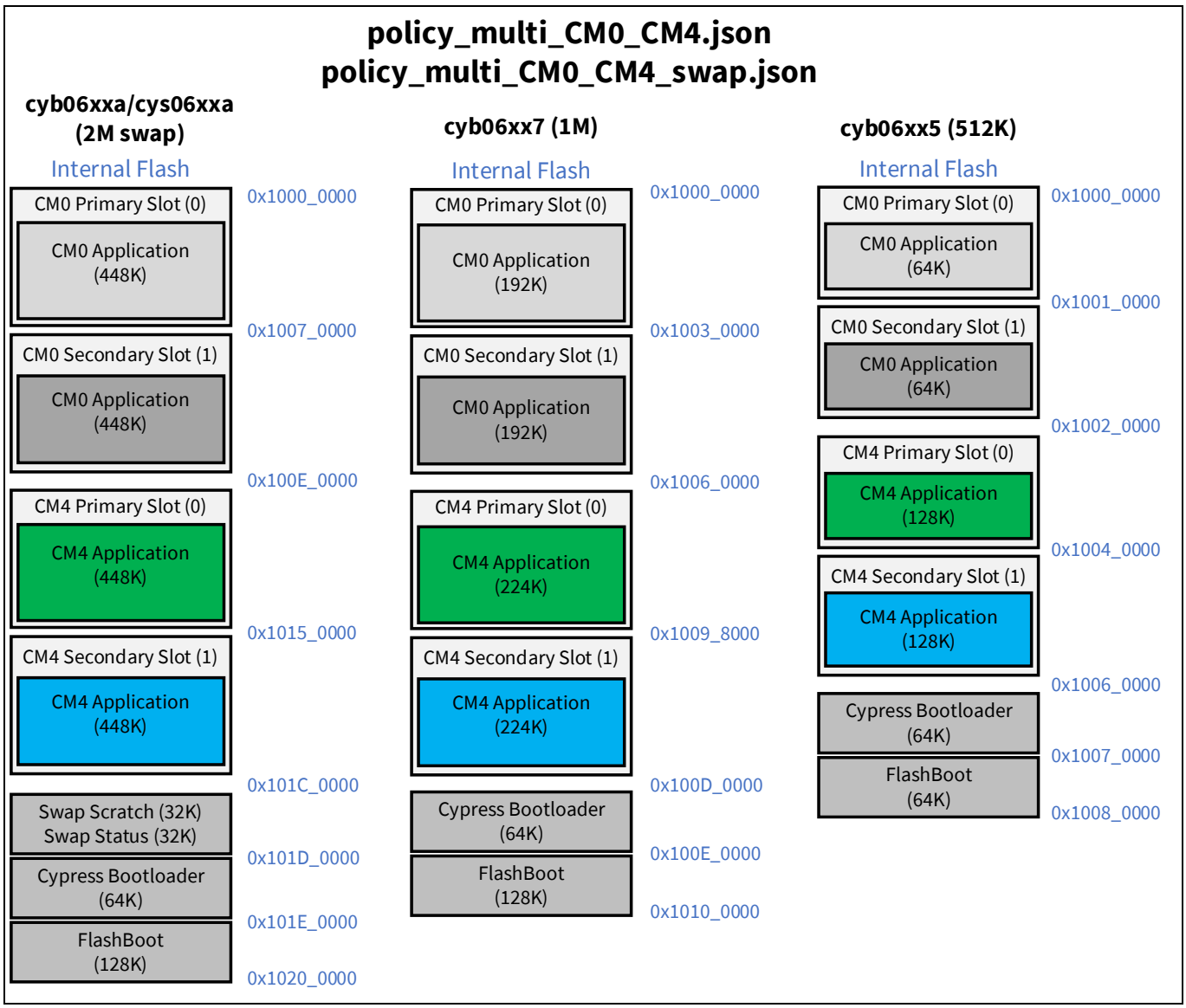

Figure 14 Flash memory map for policy_multi_CM0_CM4 example policies

6.3 Flash memory map for policy_single_CM0_CM4_smif example policies

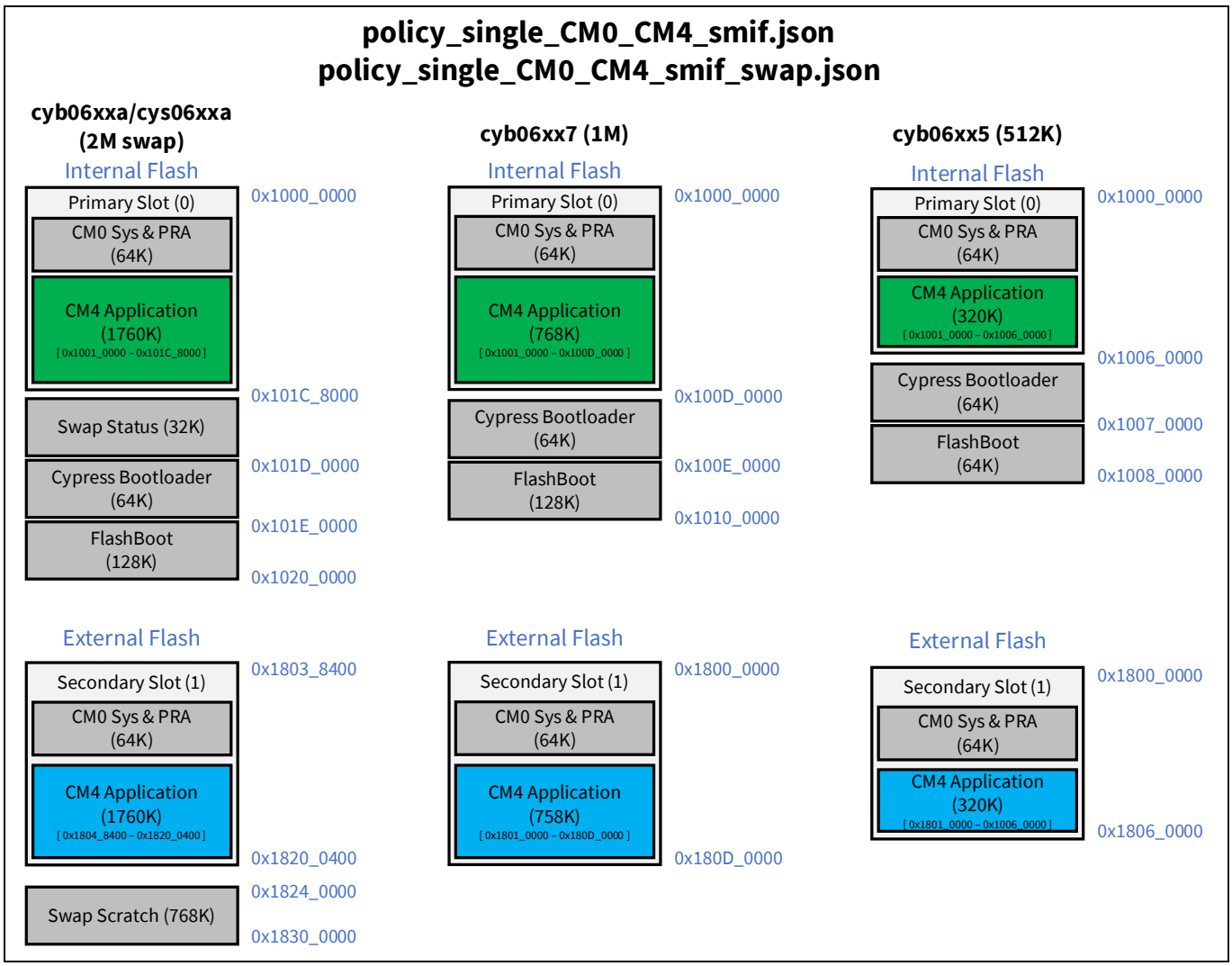

Figure 15 Flash memory map for policy_single_CM0_CM4_smif example policies

6.4 Flash memory map for policy_multi_CM0_CM4_smif example policies

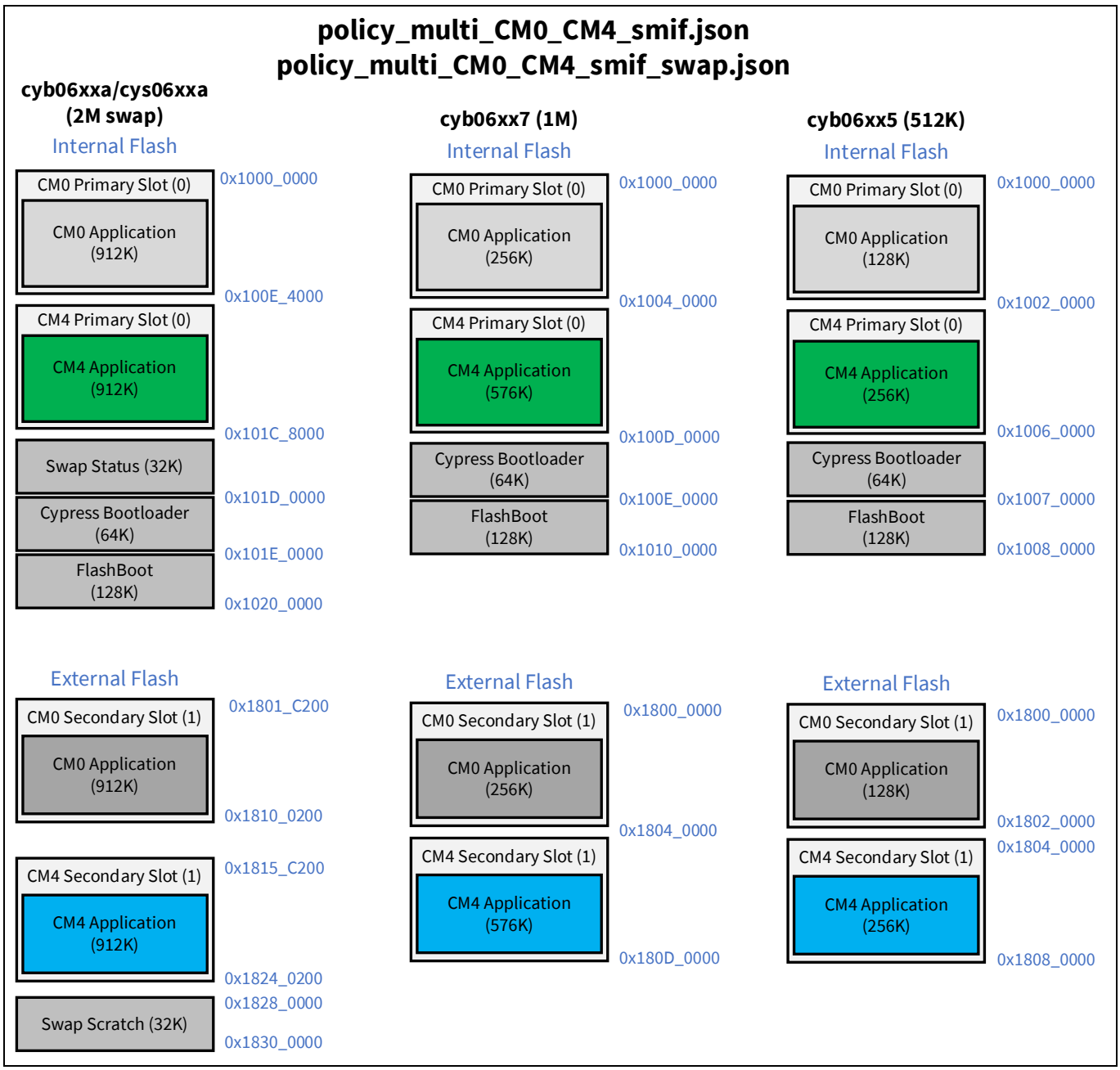

Figure 16 Flash memory map for policy_multi_CM0_CM4_smif example policies

Revision history

Document revision	Date	Description of changes
**	2019-07-19	New document.
*A	2019-09-18	Updated Mbed OS – Provisioning Flow: Updated almost entire chapter (to change to CySecureTools flow). Updated Provisioning Script Flow Details: Added “Provisioning JWT packet Reference”.
*B	2019-12-04	Updated Overview: Removed “Installing CySecureTools”. Added “CySecureTools Installation and Documentation”. Updated Mbed OS – Provisioning Flow: Updated almost entire chapter (to include ModusToolbox 2.0 flow). Added “ModusToolbox 2.0 – Provisioning Flow”. Removed “Provisioning Script Flow Details”. Added “CySecureTools Design”.
*C	2020-05-18	Updated for production silicon. Removed “Mbed OS – Provisioning Flow”.
*D	2020-07-23	Added “Mbed OS – Provisioning Flow”. General cleanup of the document.
*E	2020-12-10	Updates to include ModusToolbox 2.2 and CySecureTools flow. Fixed several typos and policy updates. Updated “CySecureTools” Design: Updated Understanding the Default Policy: Added “Policy and Configuration Limitations”.
*F	2021-04-16	Updated Document Title to read as ‘“Secure Boot”, SDK User Guide’. Updated Mbed OS path name for kit CY8CKIT064B0S2_4343W due to character limit. Updated several diagrams for correct terminology. Added more description for policy file definition. Updated provisioning procedure to sync with “CySecureTools” 3.1. Added “Additional Resources”. Added “Appendix A, Flash Memory Maps”.
*G	2023-08-09	Updated ModusToolbox™ Tools Provisioning Flow: Updated Prerequisites: Updated “CySecureTools” Installation: Updated Table 2. Removed “Mbed OS Provisioning Flow”. Migrated to Infineon template. Completing Sunset Review.

Trademarks
All referenced product or service names and trademarks are the property of their respective owners.

Edition 2023-08-09

Published by

Infineon Technologies AG

81726 Munich, Germany

Do you have a question about this document?

Email: erratum@infineon.com

Document reference

002-27860 Rev. *G