Secure Boot SDK user guide

About this document

Scope and purpose

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

Intended audience

The Secure Boot SDK is intended for PSOC™ 64 device users.

Important notice

Table of contents

1 Introduction

Infineon provides the Secure Boot SDK to simplify the use of the PSOC™ 64 Secure Boot MCU line of devices. This SDK includes all the required libraries, tools, and sample codes to provision and develop applications for PSOC™ 64 devices.

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

1.1 How to obtain the Secure Boot SDK

The main component of the SDK is a Python package called CySecureTools. It is available for download at CySecureTools.

Other components are described in later sections of this document.

1.2 Using this guide

This guide provides a high-level overview of the Secure Boot SDK, including details on the provisioning process and descriptions of the provided scripts and tools. Additionally, it offers a reference to the tokens and JSON structures used in the SDK.

This guide requires familiarity with public key cryptography, public/private key pairs, and digital signatures. An overview of these concepts is available under Public-key cryptography.

2 Overview

The PSOC™ 64 Secure Boot MCU line, based on the PSOC™ 6 MCU platform, features out-of-box security functionality. This line provides an isolated Root of Trust (RoT) with true attestation and provisioning services. Additionally, these MCUs offer a pre-configured secure execution environment that 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, first provision the device with keys and policies. Then, program the device with signed firmware. Otherwise, the device will not boot correctly. The Secure Boot SDK provides development tools to demonstrate the provisioning and signing flow.

Furthermore, the Infineon Bootloader enables Secure Boot and firmware updates.

2.1 Secure Boot SDK components

The Secure Boot SDK is organized as a standalone Python package called CySecureTools. It contains all the required scripts, default provisioning packets, and the default policy file, as shown in Table 1.

Table 1 Components and its purpose

Component	**Purpose**
Command-line tool	Allows the use of CySecureTools as a command-line utility to perform all required operations.
Provisioning scripts	Python scripts for provisioning the PSOC™ 64 Secure Boot MCU. These scripts are based on Python.
Entrance exam scripts	Runs an entrance exam on the PSOC™ 64 Secure Boot MCU to ensure that no tampering has occurred.
Infineon Bootloader image	The first-stage bootloader is based on the open-source MCUBoot [1)] library.
Sample provisioning policies	Examples to be used as templates for forming provisioning tokens.

1) MCUboot

2.2 Working of the Secure Boot SDK

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 to make that happen. This section describes the process at a high 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 an 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 a device, there is a secure 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 further security-related interactions if they are appropriately signed by the OEM.

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

To work with secure devices, the OEM provides three key components:

1. **A set of cryptographic keys**: The public key from this set is used for validating OEM application firmware
2. **A set of security policies**: These define how the secure chip should behave
3. **Certificates (optional)**: These are 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 establishes 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 in turn verifies the signature and the integrity of any code before launching it.

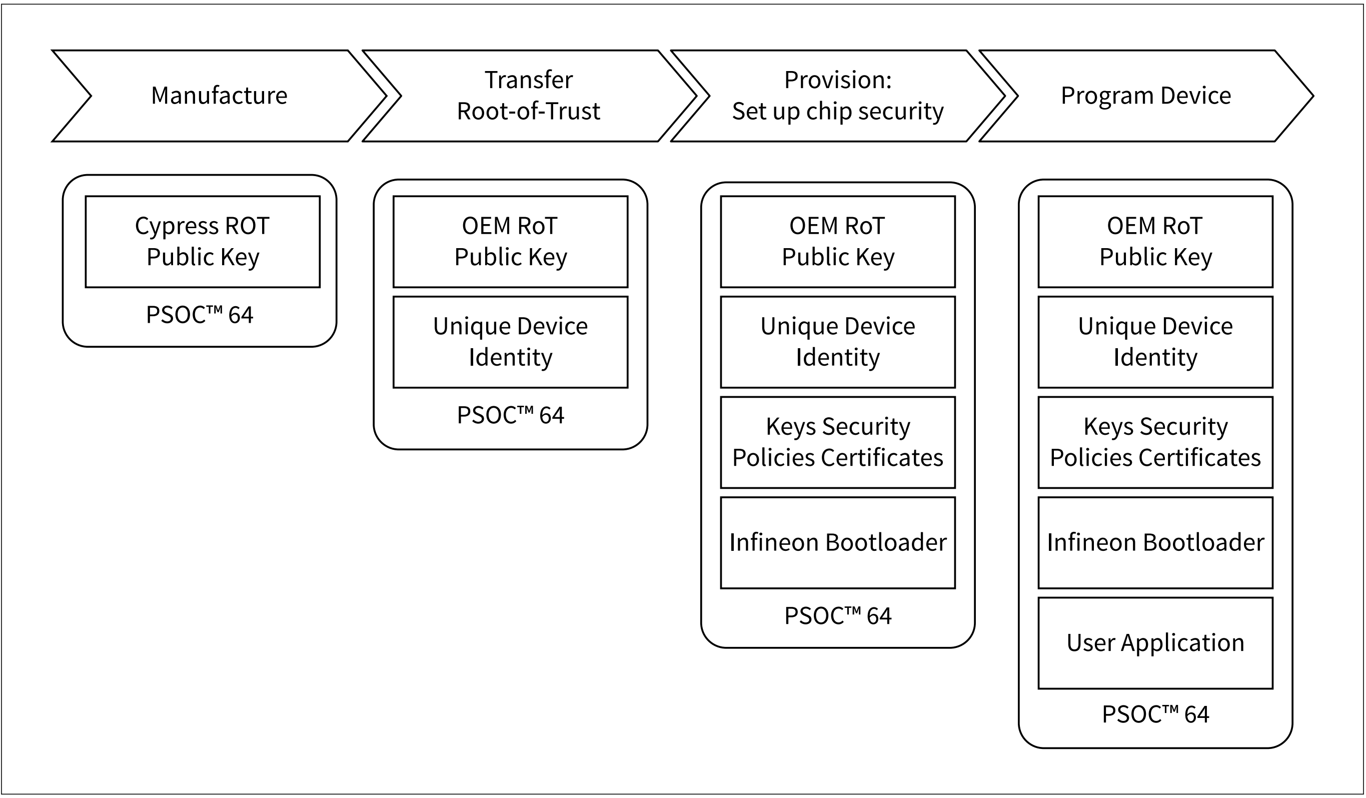

Figure 1 **PSOC™ 64 usage processes**

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 certificates
  - 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 Provisioning

Provisioning is the process of injecting secured assets like keys and security policies into the device. During development, a software team can manage this process. 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:

1. Transferring RoT: From Infineon to the development user (referred to as OEM in this document)
2. 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 an Infineon public key placed in the part during manufacturing. This Infineon public key acts as the initial Root of Trust (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:

1. **Infineon**: The owner of the Infineon Root private key
2. **Secure manufacturing environment HSM**: The entity authorized to provision and program the PSOC™ 64 Secure Boot MCU
3. **OEM/Developer**: The user or code developer of the part
4. **PSOC™ 64 Secure Boot MCU**: The holder of the Infineon Root public key

Figure 2 shows a high-level view.

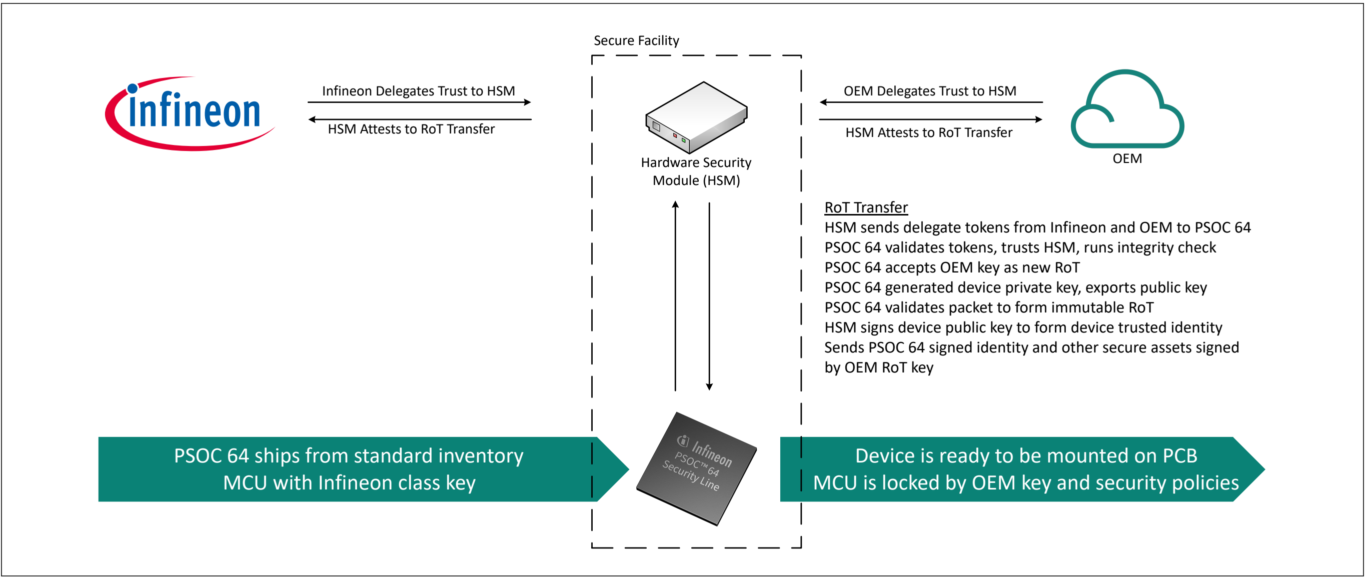
Secure Facility
Infineon Delegates Trust to HSM
HSM Attests to RoT Transfer
OEM Delegates Trust to HSM
HSM Attests to RoT Transfer
Hardware Security Module (HSM)
OEM
RoT Transfer
HSM sends delegate tokens from Infineon and OEM to PSOC 64
PSOC 64 validates tokens, trusts HSM, runs integrity check
PSOC 64 accepts OEM key as new RoT
PSOC 64 generated device private key, exports public key
PSOC 64 validates packet to form immutable RoT
HSM signs device public key to form device trusted identity
Sends PSOC 64 signed identity and other secure assets signed by OEM RoT key
PSOC 64 ships from standard inventory MCU with Infineon class key
Device is ready to be mounted on PCB MCU is locked by OEM key and security policies

Figure 2 **Infineon authoring the HSM**

The series of steps to transfer the RoT 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 result of this RoT transfer process can be represented as follows:

1. The PSOC™ 64 Secure Boot MCU now uses the OEM RoT public key as the root key to validate any OEM asset (for example, image keys, policies, etc.). This permanently and irrevocably replaces the Infineon RoT

2. 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 Infineon and the OEM authorizing an HSM is shown in Figure 3.

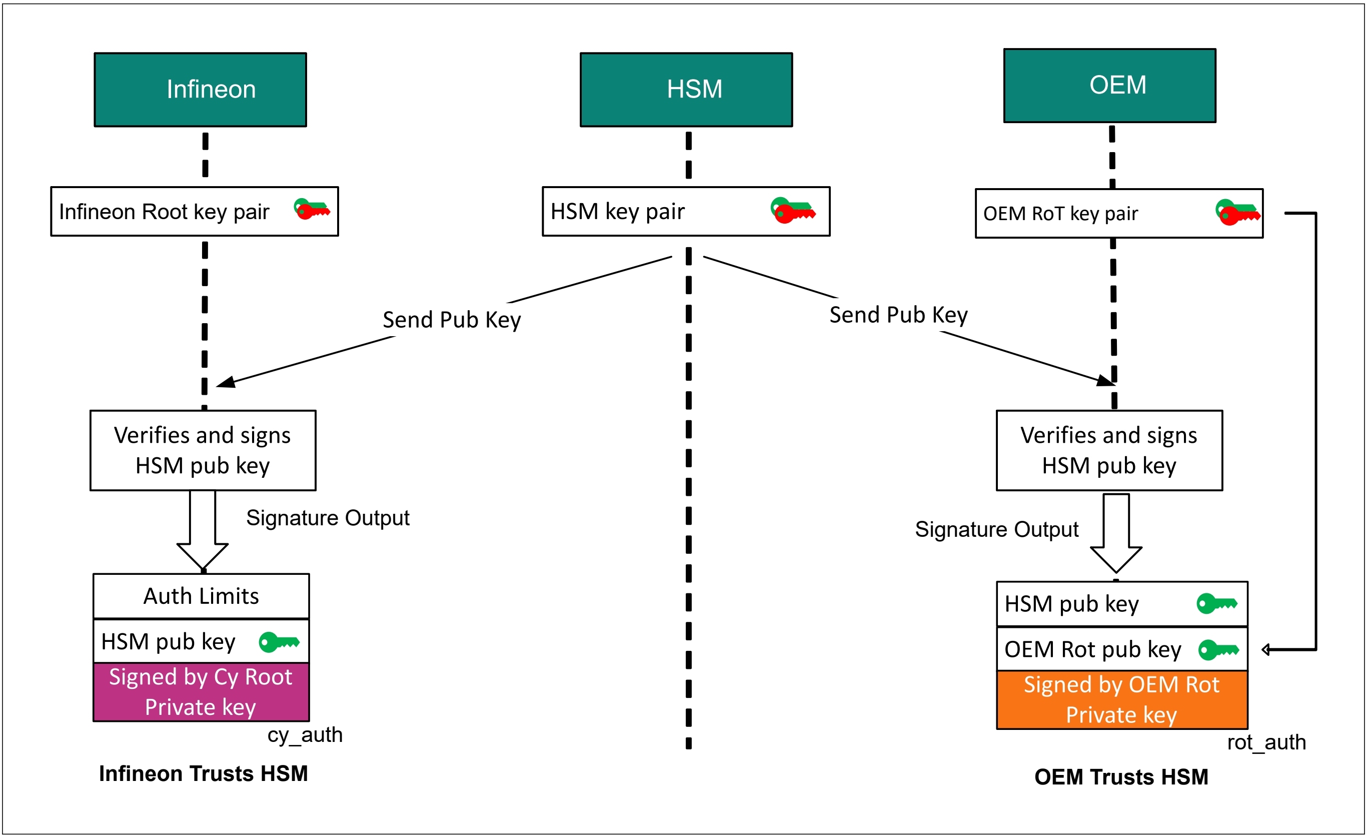

Figure 3 OEM/User authoring the HSM

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 Figure 4.

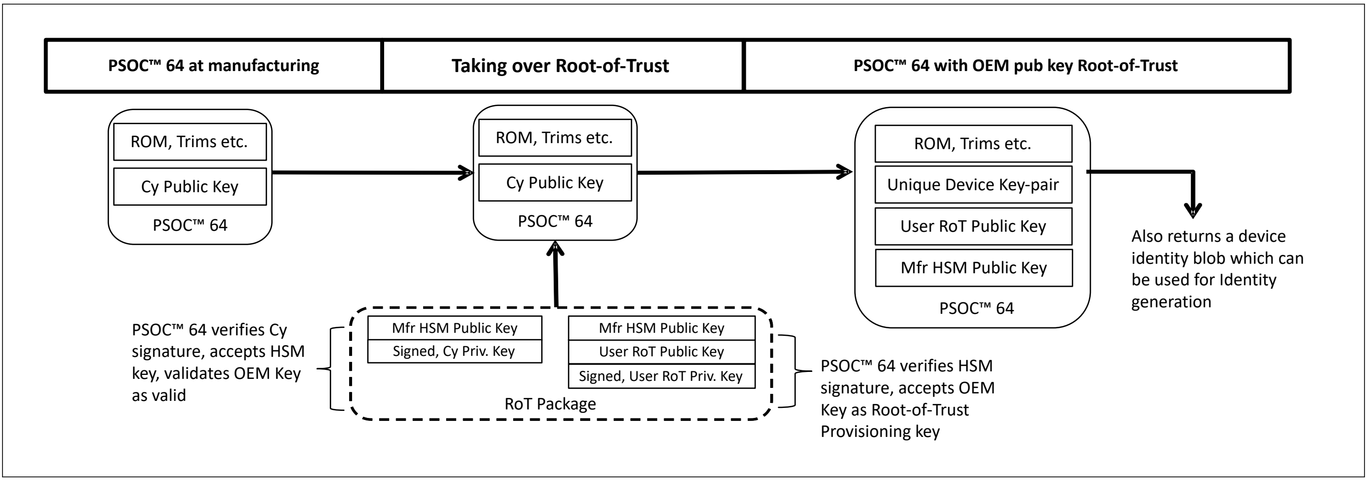
PSOC™ 64 at manufacturing
Taking over Root-of-Trust
PSOC™ 64 with OEM pub key Root-of-Trust
ROM, Trims etc.
Cy Public Key
PSOC™ 64
ROM, Trims etc.
Cy Public Key
PSOC™ 64
ROM, Trims etc.
Unique Device Key-pair
User RoT Public Key
Mfr HSM Public Key
PSOC™ 64
Also returns a device identity blob which can be used for Identity generation
PSOC™ 64 verifies Cy signature, accepts HSM key, validates OEM Key as valid
Mfr HSM Public Key
Signed, Cy Priv. Key
Mfr HSM Public Key
User RoT Public Key
Signed, User RoT Priv. Key
RoT Package
PSOC™ 64 verifies HSM signature, accepts OEM Key as Root-of-Trust Provisioning key

Figure 4 **HSM presenting the authorization to PSOC™ 64**

After the root-of-trust packet is sent, the device generates a unique device key pair and exports the generated device public key along with its unique ID. This combination can be used to link 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 and 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). It also specifies the device behavior when transitioning into RMA mode
- **Chain-of-Trust certificates**: Any certificates needed on the device; for example, a 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. The chain of trust is considered an opaque object.

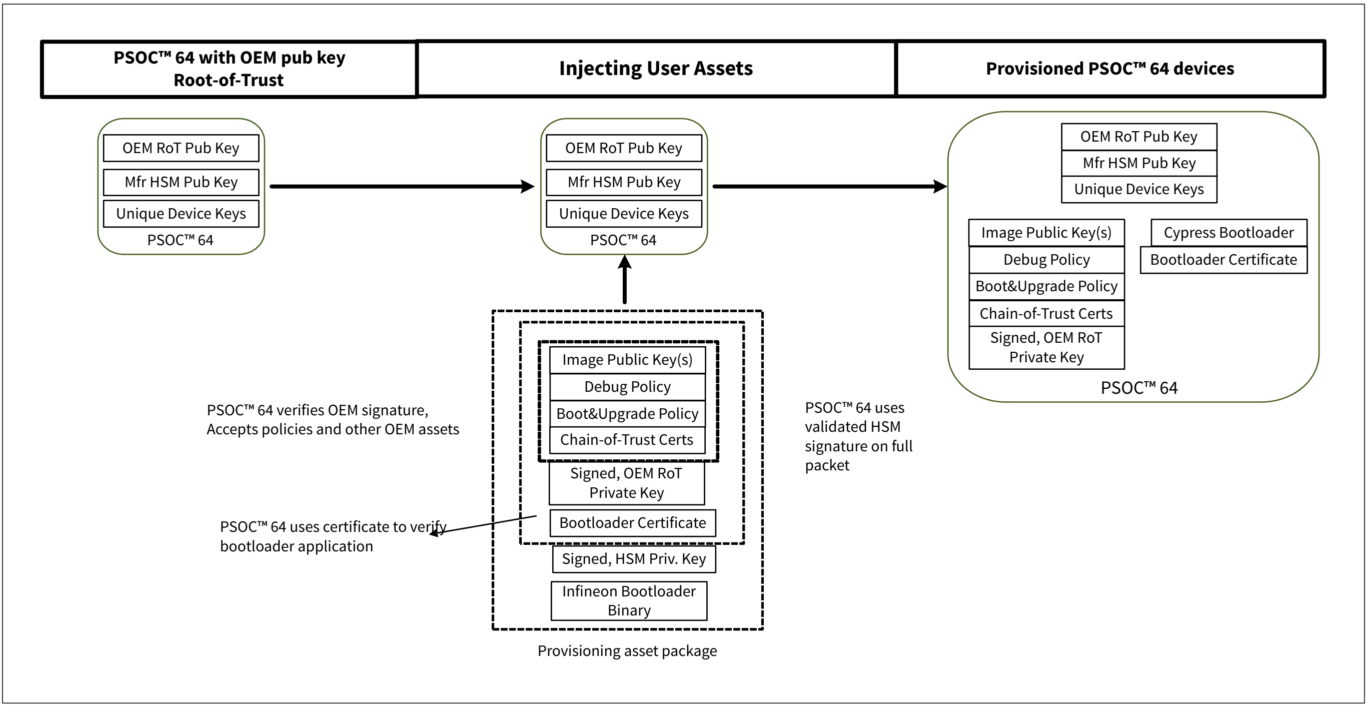
PSOC™ 64 with OEM pub key Root-of-Trust
Injecting User Assets
Provisioned PSOC™ 64 devices
OEM RoT Pub Key
Mfr HSM Pub Key
Unique Device Keys
PSOC™ 64
Image Public Key(s)
Debug Policy
Boot&Upgrade Policy
Chain-of-Trust Certs
Signed, OEM RoT Private Key
Bootloader Certificate
Signed, HSM Priv. Key
Infineon Bootloader Binary
Provisioning asset package
Cypress Bootloader
PSOC™ 64 verifies OEM signature, Accepts policies and other OEM assets
PSOC™ 64 uses certificate to verify bootloader application
PSOC™ 64 uses validated HSM signature on full packet

Figure 5 **Provisioning flow**

In addition to the OEM assets, the Infineon Bootloader is programmed at this stage, along with the Bootloader certificate (called `image_cert.JWT`) that contains the signature of the Infineon Bootloader binary. This ensures that the bootloader itself can be verified and trusted.

For more details on the Infineon Bootloader, see Infineon Bootloader.

2.3.3 Re-provisioning user assets

The PSOC™ 64 Secure Boot MCUs also allow some user assets to be re-provisioned if permitted 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 certificates

All other assets, such as the OEM RoT public key, HSM public key, and device-unique keys, cannot be replaced through 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 the open-source library MCUBoot and is capable of parsing the provisioned Boot and Upgrade policy and launching the next image if all required checks pass. For more details about this open-source library, see MCUBoot Bootloader design.

The bootloader recognizes two memory areas for each application partition: the primary slot and the 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, while 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 Single-image mode, there is one primary slot and one secondary slot since the CM0+ Secure co-processor and CM4 binaries are combined. In 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). This configuration 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 copies the secondary slot into the primary slot, then invalidates the secondary slot. The Swap method safely swaps the primary and secondary slots, allowing you to revert to the previous version of firmware if needed. After the primary slot has been updated and validated using either method, the new firmware is executed.

***Note**: The current version of the Infineon Bootloader supports image Replace upgrades for 512K and 1M flash devices and image swap for 2M flash devices. The Swap mode requires an additional 64K of internal flash if both primary and secondary slots are on-chip. If the secondary slot is external, 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 supports only external memory vendors who follow the Serial Flash Discoverable Parameters (SFDP) protocol.

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 Environment (NSPE) code need to be independently updated with individual Boot and Upgrade slots.

The Infineon Bootloader also enforces protection contexts for the bootloader code, ensuring that code running in another protection context cannot overwrite or tamper with the boot code. Figure 6 shows the launch code sequence of the Infineon Bootloader.

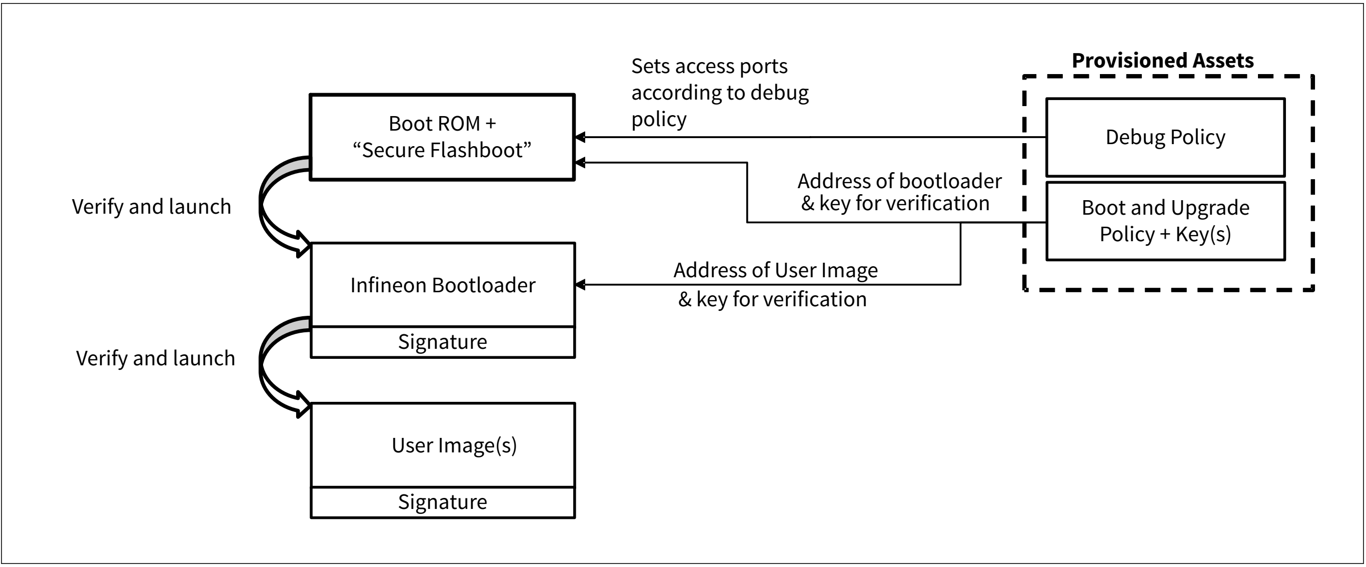
Provisioned Assets
Sets access ports according to debug policy
Boot ROM + "Secure Flashboot"
Debug Policy
Address of bootloader & key for verification
Boot and Upgrade Policy + Key(s)
Verify and launch
Infineon Bootloader
Signature
Address of User Image & key for verification
Verify and launch
User Image(s)
Signature

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 accordingly
- Checks if the boot area (primary slot) contains an image to boot
- Verifies that this image has a 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 from the primary slot if no valid image is found in the staging area

If the staging area (secondary slot) has a new image, the Infineon Bootloader performs the following operations:

1. Verifies the digital signature of the image located in the secondary slot
2. Decrypts the image's body and verifies the digital signature of the decrypted image (optional for encrypted image support)
3. Checks that the corresponding policies allow the upgrade
4. Checks that the image metadata matches the image in the primary slot, then upgrades it

- **Replace mode**:
  - Overwrites the primary slot with the decrypted (if needed) image from the secondary slot
  - Invalidates the 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
  - Re-encrypts the secondary slot if it had been encrypted previously

Figure 7 shows a typical application update scenario using the Infineon Bootloader.

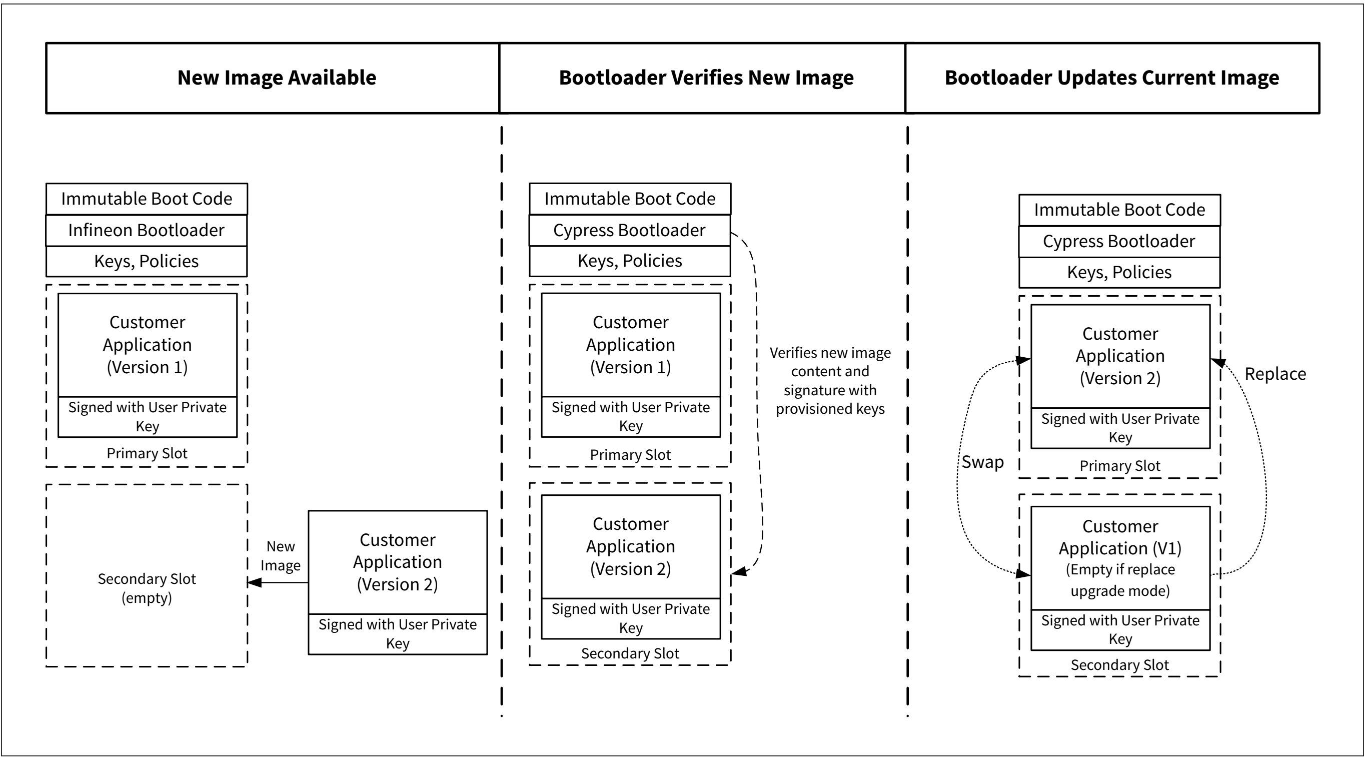
New Image Available
Bootloader Verifies New Image
Bootloader Updates Current Image
Immutable Boot Code
Infineon Bootloader
Keys, Policies
Customer Application (Version 1)
Signed with User Private Key
Primary Slot
Secondary Slot (empty)
New Image
Customer Application (Version 2)
Signed with User Private Key
Immutable Boot Code
Cypress Bootloader
Keys, Policies
Customer Application (Version 1)
Signed with User Private Key
Primary Slot
Customer Application (Version 2)
Signed with User Private Key
Secondary Slot
Verifies new image content and signature with provisioned keys
Immutable Boot Code
Cypress Bootloader
Keys, Policies
Customer Application (Version 2)
Signed with User Private Key
Primary Slot
Customer Application (V1) (Empty if replace upgrade mode)
Signed with User Private Key
Secondary Slot
Swap
Replace

Figure 7 **Bootloader application update sequence**

2.5 CySecureTools installation and documentation

The stand-alone Python package CySecureTools includes 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 Python and requires interpreter versions higher than 3.7.

The source code for CySecureTools is available on GitHub. Full details about the operations, commands, and APIs can be found in the `README.md` file.

To install and configure CySecureTools:

1. Install Python 3.8.10 or later on your computer. Download it from the Python website or install it using the package manager of your host system

2. Set up the appropriate environment variables for your operating system

3. If Python 2.7 is also installed, ensure that `Python38` and `Python38\Scripts` have higher priority in the `PATH` than Python27

Windows: To configure the `PATH` environment variable on Windows, follow these steps:

1. Open **Control Panel**

2. Go to **System > Advanced System Settings > Environment Variables**

3. Find **`PATH`** in the list of user variables

4. Click **Edit**

5. Move `C:\Python38` and `C:\Python38\Scripts` to the top

6. Save changes and exit Control Panel

7. Open a `command-line or terminal` program and run the following command to verify the Python version:

```
python -version
```

Linux: To configure Python on Linux, follow these steps:

Most distributions of Linux should already have Python 2 and Python 3 installed. To verify that Python by default points to Python 3, run:

```
python --version
```

If Python 3 is not set as the default, run the following commands. The number at the end of each command denotes priority:

```
update-alternatives --install/usr/bin/python python/usr/bin/python3.8 1
                     update-alternatives --install/usr/bin/python python/usr/bin/python2.7 2
```

macOS: To configure Python on macOS, follow these steps:

By default, Python points to `/usr/bin/python`, which is Python 2. To make Python and Pip resolve to the Python 3 versions, execute the following commands from the command line:

```
echo 'alias python=python3' >> ~/.bash_profile
echo 'alias pip=pip3' >> ~/.bash_profile
source ~/.bash_profile
python --version
Python 3.8.10
pip --version
pip 23.3.2 from /Library/Frameworks/Python.framework/Versions/3.8/lib/python3.8/site-
packages/pip (python 3.8)
```

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

To install the CySecureTools package, first ensure that you have the latest version of pip installed. Use the following command:

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

To install the CySecureTools package, which is 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 errors can be safely ignored.*

To show the path to the installed package, use the following command:

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

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

The following are the instructions for the currently recommended version:

For Windows:

1. Download libusb from libusb.info
2. Place the DLL file in your Python installation folder, next to `python.exe`
3. Ensure that you use the same 32- or 64-bit architecture as your Python installation

***Note**:* *Due to a known issue, the current recommendation is to use libusb version 1.0.21 instead of the most recent version.*

For Linux:

Use the host system's package manager to install the driver using a terminal. Note that this command requires `sudo` privileges. For example, to install the driver on Ubuntu, run the following command:

```
apt-get install libusb
```

For macOS:

Use the Homebrew package manager to install the driver using the terminal. Run the following command:

```
homebrew install libusb
```

3 ModusToolbox™ tool 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 ModusToolbox™ software, version 3.0 or later. See the ModusToolbox™ installation guide for instructions.

***Notes**:*

1. *On Windows, note that Python is no longer bundled with ModusToolbox™ 3.2 or later. In most cases, install Python version 3.8.10 to 3.11. See KBA239118 - Using Python with a ModusToolbox™ application to set up Python with ModusToolbox™ 3.2. This content has been tested with ModusToolbox™ 3.2 and Python 3.8.10*
2. *After installing ModusToolbox™ software on Linux machines, run the `ModusToolbox/tools_<version>/ modus-shell/postinstall` script*
3. *ModusToolbox™ software provides a Secure Policy Configurator with a graphical user interface to modify policy tools and provision the device. However, this Secure Boot SDK user guide uses the command-line option, as it defaults to a standard policy file that does not require modification for basic kit and device evaluation*

3.1.2 CySecureTools installation

Follow the instructions in the CySecureTools installation and documentation section. This example will use the CY8CKIT-064B0S2-4343W kit, although you are free to use any of the PSOC™ 64 kits. The target parameter for CySecureTools can be either the kit name or the device family name. See Table 2 for reference. 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 equivalent:

```
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 Creating a Secure Blinky LED FreeRTOS 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 **Next >**
5. Select the **Secure Blinky LED FreeRTOS** application and click **Create**. This may take a while as it pulls all required sources from the 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. A pre-signed development token is available in the SDK, which authorizes an HSM key-pair provided in the SDK.

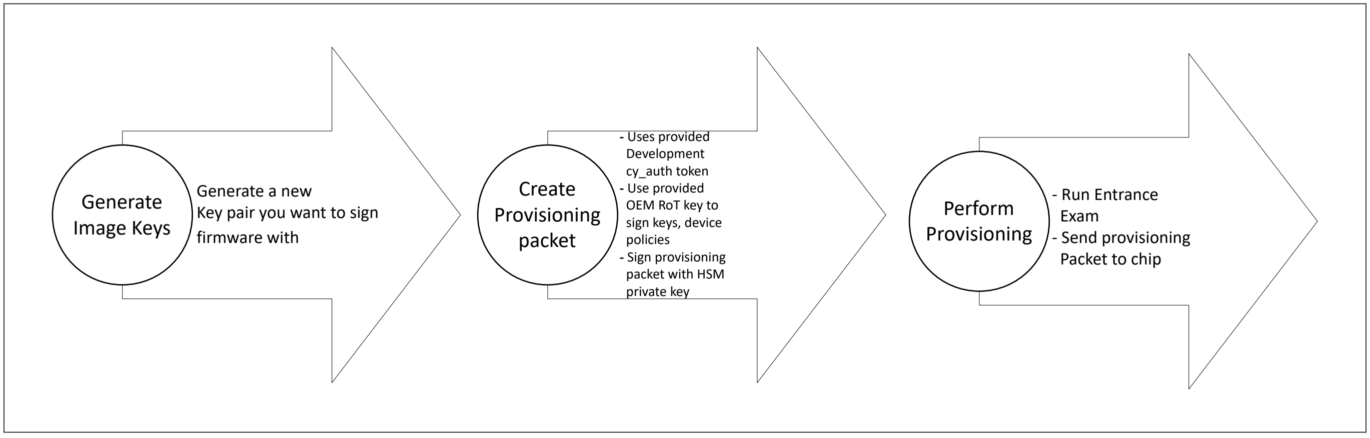
Generate Image Keys
Generate a new Key pair you want to sign firmware with
Create Provisioning packet
- Uses provided Development cy_auth token
- Use provided OEM RoT key to sign keys, device policies
- Sign provisioning packet with HSM private key
Perform Provisioning
- Run Entrance Exam
- Send provisioning Packet to chip

Figure 8 **Provisioning flow**

All the following steps mentioned in the subsequent subsections should be executed from the command line. The path to the policy file (if using a custom policy) can be relative to the current working directory or absolute. All paths to key files inside the policy file must be absolute or relative to the policy path.

3.2.1 A. Set up the CySecureTools workspace

To provision the CY8CKIT-064B0S2-4343W kit using CySecureTools, follow these steps:

1. **Open a command-line application**:
   - Open a native command-line application and navigate to the `%WORKSPACE%/Secure_Blinky_LED_FreeRTOS/` directory
   - For Windows, use the "modus-shell" program provided in the ModusToolbox™ installation instead of a standard Windows command-line application. This shell provides access to all ModusToolbox™ tools. Access modus-shell by typing "modus-shell" in the Windows search box in the Windows menu
2. **Provisioning example**:
   - The following provisioning example uses the CY8CKIT-064B0S2-4343W kit as the target. 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 in all the following commands)

```
cysecuretools -t cyb06xxa init
```

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.

***Notes**:*

1. *After running this step, you will have a choice of multiple default policies you can use to provision the chip. Choose which policy you want to use by using the `--policy/p` flag in the CySecureTools CLI. Ensure you use the same policy file when running through steps B, C, and D*
2. *If you are not using the CY8CKIT-064B0S2-4343W kit, the default policy file name will be `policy_single_CM0_CM4.json`. See the `policy` directory to examine the example `policy` files*

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

3.2.2 B. Generate new keys

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

In your command line, copy and paste the following command:

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

CySecureTools reads the provided policy and generates the keys defined within it. Depending on the chosen policy, multiple keys can be 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 this step. Press the Mode button on the kit until the Status LED blinks rapidly. For more details, see the** KitProg3 user guide.*

In your command line, copy and paste the following command:

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

The entrance exam is a test routine that:

- Verifies that the device is in the correct lifecycle stage
- Verifies that the boot code has not been modified or tampered with
- Verifies 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 provide an option to erase the chip. Existing firmware can be erased using tools like Infineon Programmer.

***Note**:* *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 and paste the following command for device provisioning:

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

CySecureTools provision-device API performs 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 do not need to modify it. See Understanding the default policy for more details.

3.3 Device re-provisioning

The default device policy templates provided in CySecureTools allow re-provisioning of a device after running through the initial 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 during re-provisioning, see Re-provisioning after failure.

3.4 ModusToolbox™ secure image generation

In ModusToolbox™, 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 directory; for example, `..\mtb_shared\ \TARGET_CY8CKIT-064B0S2-4343w\latest-v4.X\CY8CKIT-064B0S2-4343W.mk`.

Figure 9 shows the flow for signing and encryption using ModusToolbox™ tools.

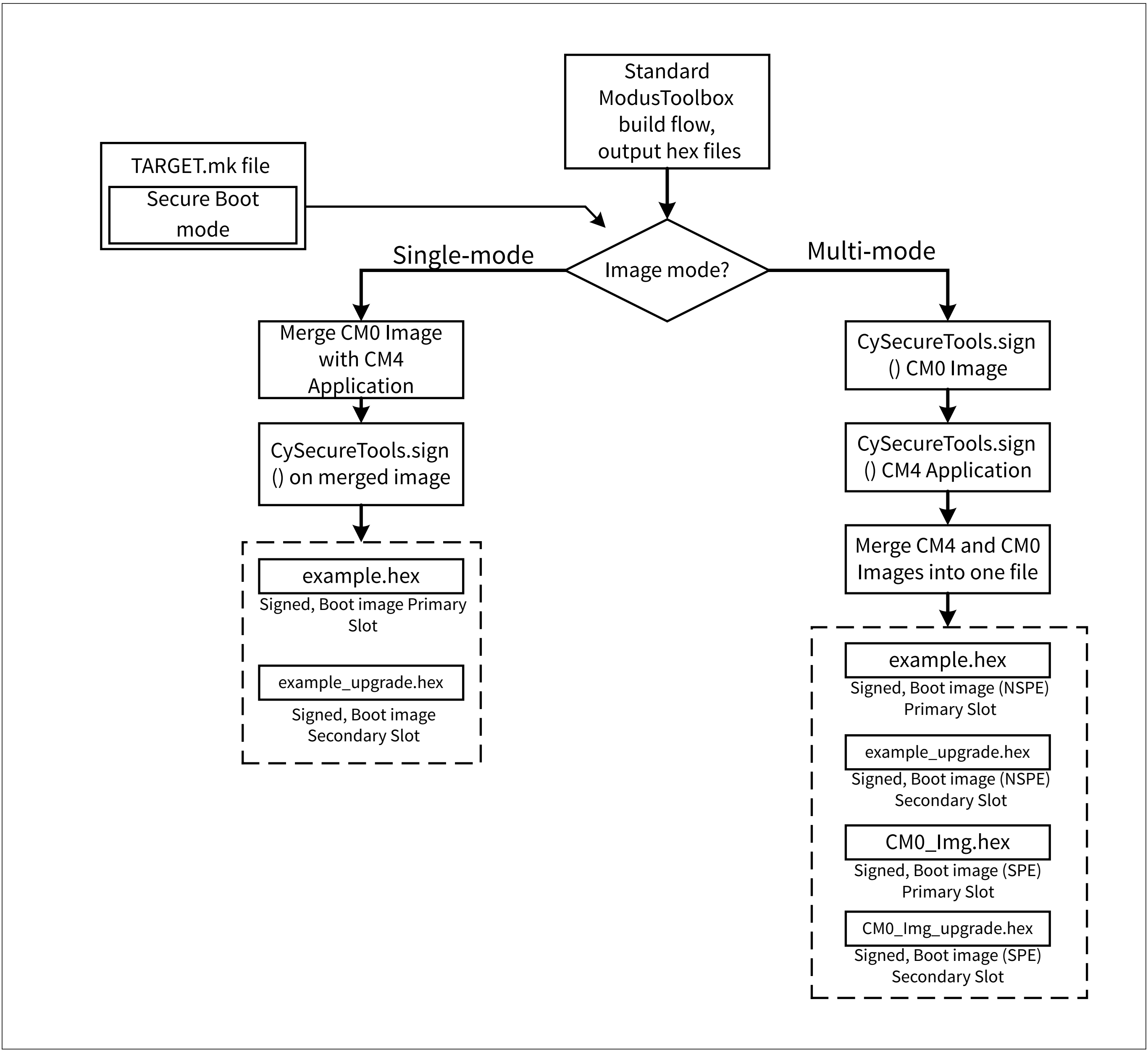
Standard ModusToolbox build flow, output hex files
TARGET.mk file
Secure Boot mode
Image mode?
Single-mode
Multi-mode
Merge CM0 Image with CM4 Application
CySecureTools.sign() on merged image
example.hex
Signed, Boot image Primary Slot
example_upgrade.hex
Signed, Boot image Secondary Slot
CySecureTools.sign() CM0 Image
CySecureTools.sign() CM4 Application
Merge CM4 and CM0 Images into one file
example.hex
Signed, Boot image (NSPE) Primary Slot
example_upgrade.hex
Signed, Boot image (NSPE) Secondary Slot
CM0_Img.hex
Signed, Boot image (SPE) Primary Slot
CM0_Img_upgrade.hex
Signed, Boot image (SPE) Secondary Slot

Figure 9 **Secure image generation**

The build process outputs two binaries:

1. **Signed boot image Hex file**: This binary can be programmed to the primary slot of the PSOC™ 64 device to securely launch the application
2. **Signed and encrypted (policy dependent) Update image Hex file**: This binary can be programmed to the secondary slot of the PSOC™ 64 device to perform a secure update and then launch the application

3.5 Build and run the application

If you have just completed the provisioning process, the MiniProg4 or the kit/protoboard KitProg might still be in DAPLink mode, which is required for provisioning. Change the MiniProg4 or KitProg back to CMSIS-DAP Bulk mode before attempting to program or debug the device. The MiniProg4 or KitProg status LED should be ON but not blinking to indicate CMSIS-DAP Bulk mode. To return the device to this mode, press the Mode Select button on the MiniProg4 or KitProg.

Also, if the supply voltage was changed to 2.5 V for provisioning, revert the supply voltage to the normal operating level before programming and application operation.

Follow these steps to build and run the project:

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

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

3.7 Re-provisioning after failure

Perform this step as needed for the following scenario:

- 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
- Attempt 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 faulty application during the 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 a timeout

To address this failure, manually erase the boot slot 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 puts 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 offers a command-line interface over stand-alone scripts that simplifies calling them with a minimum number of arguments. Advanced users can use the scripts directly without the wrapper and configure each argument as needed. A graphical user interface (GUI) for CySecureTools is also available. See the ModusToolbox™ Secure Policy Configurator user guide, which is available with the ModusToolbox™ Edge Protect Security Suite package. For installation and more details, see the ModusToolbox™ Setup program user guide. The GUI provides both easy execution of CySecureTools without using a command line and the ability to edit the policy file without directly editing the `XML policy` file.

4.1 CySecureTools component diagram

Figure 10 shows the high-level components of CySecureTools.

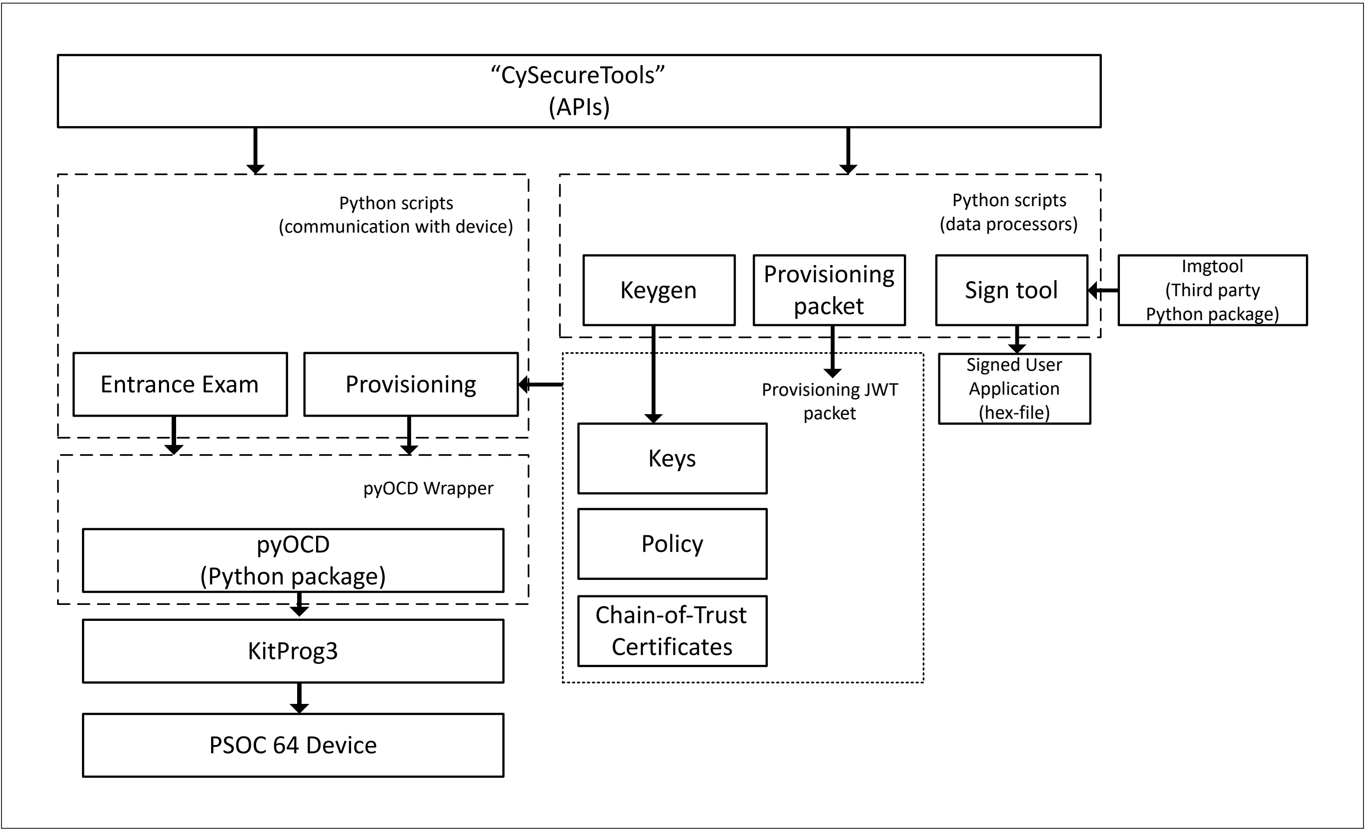

Figure 10 CySecureTools components

4.1.1 Creating a provisioning packet

The final `prov_cmd.jwt` file required for provisioning a secured MCU device needs several pieces of information. As input arguments, the tools (specifically the create-provisioning-packet API) take:

- OEM key file
- HSM key file
- Infineon Bootloader image certificate
- Provisioning authorization certificate
- Policy file
- Output directory (default is packet)

- User keys 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 to assign an 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 both provisioning and re-provisioning. This packet contains data that can be changed during re-provisioning.

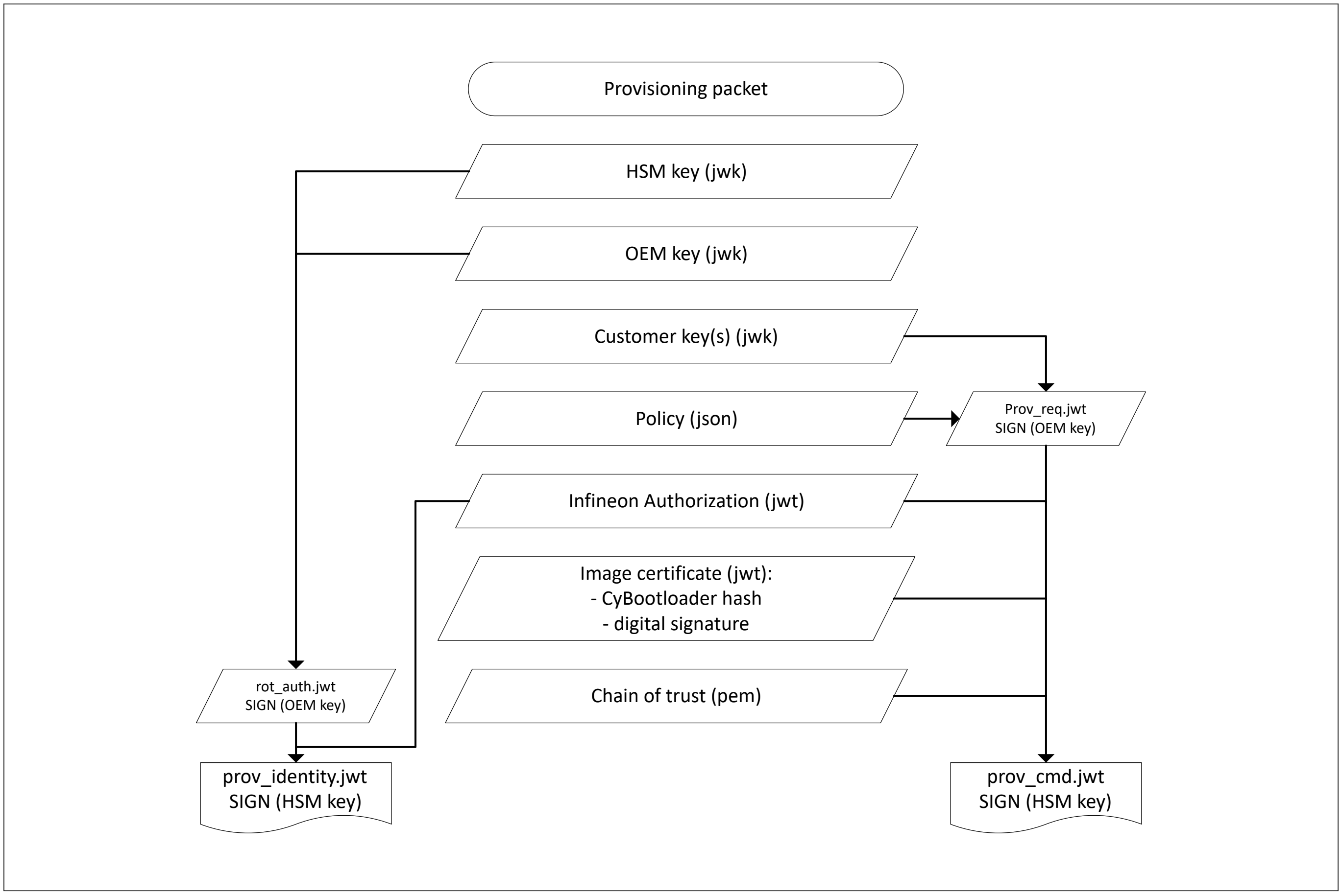
Provisioning packet
HSM key (jwk)
OEM key (jwk)
Customer key(s) (jwk)
Policy (json)
Prov_req.jwt SIGN (OEM key)
Infineon Authorization (jwt)
Image certificate (jwt): - CyBootloader hash - digital signature
Chain of trust (pem)
rot_auth.jwt SIGN (OEM key)
prov_identity.jwt SIGN (HSM key)
prov_cmd.jwt SIGN (HSM key)

Figure 11 Provisioning packet structure

4.2 Understanding the default policy

There are four example policies provided in CySecureTools for each target. 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. Table 3 shows the difference 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 ModusToolbox™)
	This policy is designed for applications that require a single signature for two combined applications: Secure CM0p and User App on CM4.
	It is typically used in 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 the previous policy, this policy supports applications requiring a single signature for two combined applications: Secure CM0p and User App on CM4. Additionally, it enables external memory support for the 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`	This policy is designed for applications that require independently updateable Secure Processing Environment (SPE) and Non-Secure Processing Environment (NSPE) code. It is typically used in IoT systems that maintain secure 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 the previous policy. Additionally, it enables external memory support for the 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 CySecureTools. The contents are classified as follows:

- Boot and Upgrade policy
- Debug policy
- Infineon Bootloader
- CySecureTools miscellaneous assets

4.2.1 Policy and configuration limitations

The following is a list of limitations with the policy and configuration of the PSOC™ 64 Secure Boot MCU:

- Only the upgrade (secondary slot) may reside in external memory (SMIF-based). The boot (primary slot) must reside in internal flash
- Only one SMIF slave-select can be used in a system with firmware upgrades enabled. Other SMIF devices with additional slave-selects can be used for non-code-related storage
- The start address of secondary slots in external memory must be 0x1800_0000 + (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, if the external memory starts at address 0x1800_0000 and the sector size is 0x40000, then the start address can be 0x1800_0000, 0x1804_0000, 0x1808_0000, etc
- Only serial flash discoverable parameter (SFDP) compatible devices are supported for external memory
- The larger sector size must be an even multiple of the smaller sector size. In 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 to or larger than the sum of the image size (an 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 three 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, while devices with less than 2M Flash will use REPLACE Upgrade mode

4.2.2 Boot and Upgrade policy

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

- Infineon BnU policy
- CM0+ Secure co-processor image BnU policy
- CM4 image BnU policy
- Re-provisioning options

Table 4 to Table 7 show the Infineon Bootloader settings in the example or default `policy_single_CM0_CM4_swap.json` file for the cyb06xxa family (2M Flash) devices. This example policy file is part of CySecureTools version 3.1. These fields may need to be modified but can be useful as a reference.

Figure 12 shows the memory map of the flash defined by the policy file. See Appendix A: Flash memory maps for memory map files of the example policy files for CySecureTools 3.1.

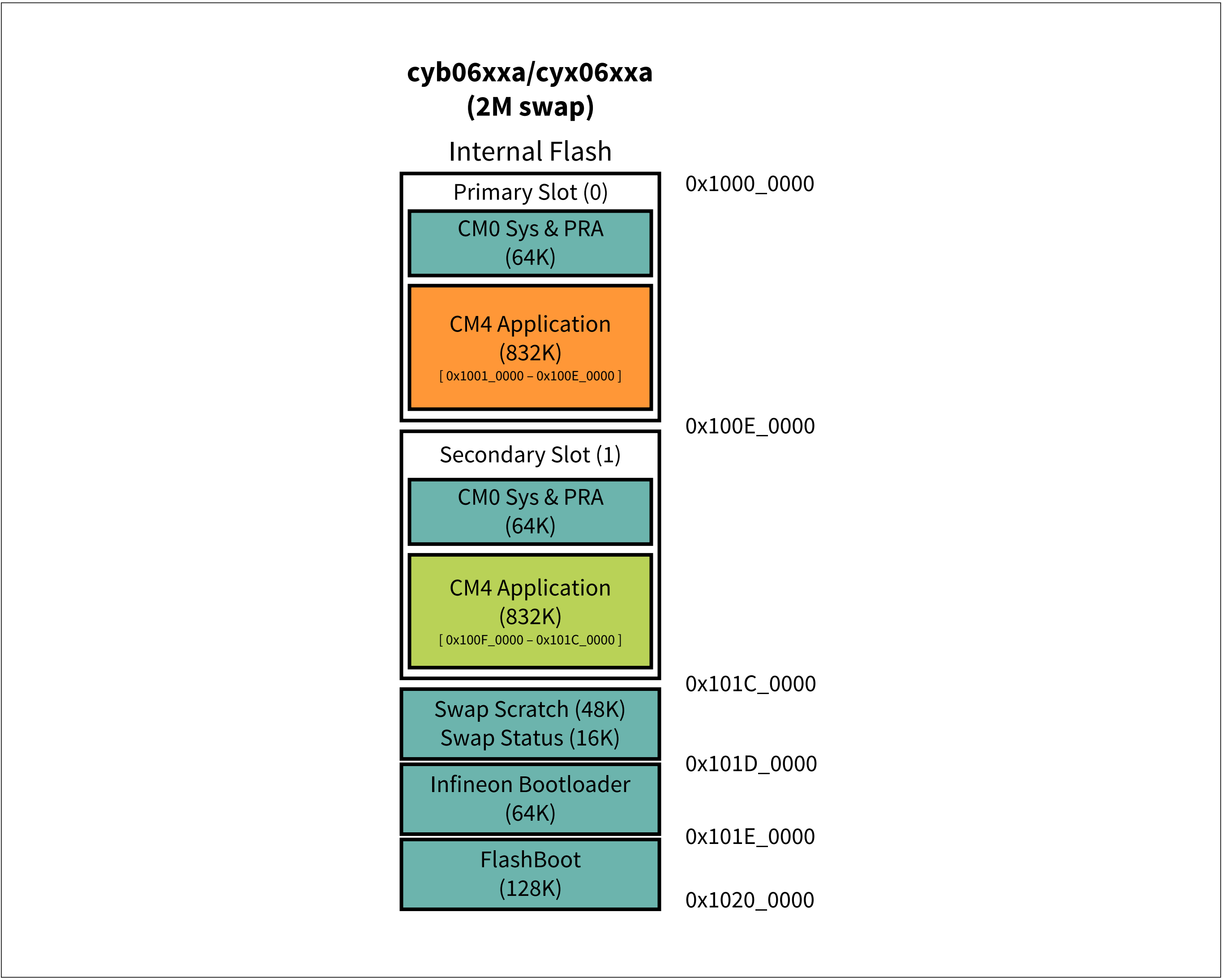

Figure 12 **PSOC™ 64 Secure MCU flash memory map**

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

JSON field	Description
{	-
"boot_auth": [5],	KeyID 5 is used to check the image signature
"bootloader_keys": [	Defines key used for the 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

(table continues...)

Table 4 (continued) Infineon Bootloader Boot and Upgrade policy

JSON field	Description
"upgrade_mode": "swap",	This device uses Swap mode for firmware upgrades. The 512K and 1M flash devices use Replace mode, which is the default if this policy is not specified.
"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.

(table continues...)

Table 4 (continued) Infineon Bootloader Boot and Upgrade policy

JSON field	Description
"address": 270319616	0x0x101C_C000
"size" 16384	0x4000 (16K)
}	-
{	-
"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 and Upgrade policy

JSON field	Description
{	-
"boot_auth": [8],	KeyID 8 used to check image signature.
"boot_keys": [	Defines keys 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).

(table continues...)

Table 5 (continued) CM0+ Secure co-processor application image Boot and Upgrade policy

JSON field	Description
"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.
"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
}	-
]	-
},	-

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

Table 6 CM4 application image Boot and 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)
}	-
],	-

(table continues...)

Table 6 (continued) CM4 application image Boot and Upgrade policy

JSON field	Description
"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)
"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 Re-provisioning 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

(table continues...)

Table 8 (continued) Infineon Debug policy

JSON field	Description
"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)
"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 an 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
"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.
"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 structure of this JWT is shown in the following code block.

```
{
  {
  "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 parameter description**

Object	Description
`cy_auth`	Infineon authorization JWT: Authorizes the HSM public key.
`rot_auth`	OEM/User authorization JWT: Authorizes the HSM public key.
`image_cert`	Infineon Bootloader image JWT: Used for sending an Infineon Bootloader signature.
`chain_of_trust`	Holds an array of X.509 certificates
`complete`	True - indicates that the complete provisioning process must be finished and the chip lifecycle should move forward.
`type`	Specifies the JWT type as a string.

4.3.2 prov_identity.jwt

The `prov_identity.jwt` is the initial token sent to the chip to create a unique identity. The structure is shown in the following code block:

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

Table 12 **prov_identity.jwt parameter description**

Object	Description
`create_identity`	If true, the 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

The structure of the `cy_auth.jwt` is shown in the following code block:

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

Table 13 cy_auth.jwt parameter 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

The structure of the `rot_auth.jwt` is shown in the following code block:

```
{
  "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 parameter 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

The structure of the `prov_req.jwt` is shown in the following code block:

```
{
  "custom_pub_key": [{Key1}, …],
  "boot_upgrade": {…},
  "debug": {…}
  "prod_id": "my_thing",
  "wounding": {}
} sig: OEM_RoT_PRIV_KEY
```

Table 15 prov_req.jwt parameter 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

The structure of the `boot_upgrade.JSON` is shown in the following code block:

```
{
    "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 parameter description

Object	Description	Range of valid values
`id`	Image id. (0-16: Infineon reserved, >16: customer specific)	A range of integers can be specified: • 0: The first firmware image started from RomBoot/ FlashBoot (that is the bootloader) • 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

(table continues...)

Table 16 (continued) boot_upgrade.JSON parameter description

Object	Description	Range of valid values
`boot_auth`	Specifies the key index to use for validating the signature. These signatures are all verified during boot.	• Can be any integer public key greater than 3 • For Infineon Bootloader, the auth is 3 • For the M4 image, this can be any number depending on the `key_id` specified in the JWK format in the `custom_pub_key` fields
`launch`	Specifies the next image ID is 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 an upgrade, this counter is incremented to the value from the image header of the upgrade image.	0 to 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, for example, 268435456 (which corresponds to 0x10000000).
`resources: size`	Specifies the size of the image.	The valid flash range size in bytes. Only decimal values are allowed, for example, 327680 (which corresponds to 0x50000 or 320 KB).
`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 -> Upgrades are not allowed.

(table continues...)

Table 16 (continued) boot_upgrade.JSON parameter description

Object	Description	Range of valid values
`upgrade_auth`	Specifies the 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 greater than 3. • For Infineon Bootloader, the auth is 3 • For the M4 image, this can be any number depending on the `key_id` specified in the JWK format in the `custom_pub_key` fields

4.3.7 debug.JSON

The structure of the `debug.JSON` is shown in the following code block:

```
{
    "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 parameter 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.

(table continues...)

Table 17 (continued) debug.JSON parameter description

Object	Description	Range of valid values
`m0p/m4/system: control`	Specifies how the DAP port can be opened after bootup. This field is only valid if permission is allowed.	Firmware: User code can choose to open the DAP port based on custom logic. 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 the control field	The key ID must be greater than 3 and point to the key provisioned in the `custom_pub_key` field.
`system: flashr/flashw`	Specifies which regions the SysAP port is allowed to access.	rue: Flash reads/writes via SysAP are allowed. false: Flash reads/writes via SysAP are 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 a key greater than 3 to a SysCall API. The system will destroy fuse and flash contents as specified in destroy_fuses and destroy_flash before transitioning to the RMA stage.
`rma: destroy_fuses: start`	Starting fuse bit number for the region.	0~65536. Check the part datasheet for the eFuse allowed address.
`rma: destroy_fuses: size`	Number of fuse bits in the region.	0~65536. Check the part datasheet for the eFuse allowed size.
`rma: destroy_flash: start`	Starting byte address of the region (will be rounded down to the nearest program/erase boundary).	0~0xFFFFFFFF. Check the part datasheet for the flash allowed address.
`rma: destroy_flash: size`	Size in bytes of the region (will be rounded up so the region is an 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 the RMA stage.	The key ID must be greater than 3 and point to the key provisioned in the `custom_pub_key` field.

5 Additional resources

This chapter includes various links to additional resources that are useful when working with the Secure Boot SDK.

Application notes:

- 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

Code examples:

- Using ModusToolbox™

Device documentation:

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

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

PSOC™ 6 Middleware (on GitHub):

- GitHub page for PSOC™ 6 middleware

Tools:

- ModusToolbox™ software
- CySecureTools page on pypi.org

5.1 Libraries (on GitHub)

Table 18 includes various links to libraries on GitHub that are useful when working with the Secure Boot SDK.

Table 18 Libraries (on GitHub)

Name	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-I/O - 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

6 Appendix A: Flash memory maps

6.1 Flash memory map for policy_single_CM0_CM4 policy files

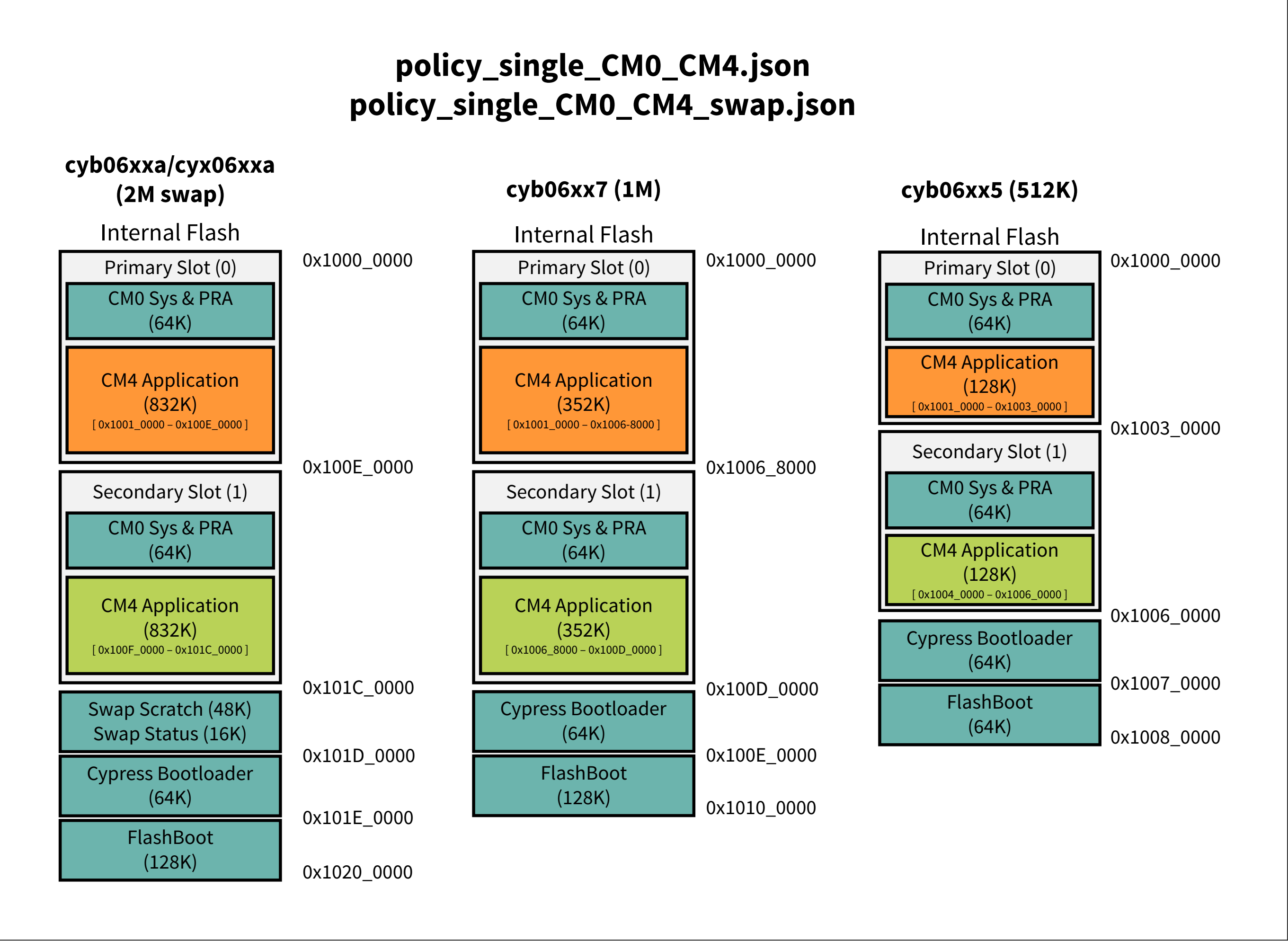

Figure 13 **Flash memory map for policy_single_CM0_CM4 policy files**

6.2 Flash memory map for policy_multi_CM0_CM4 example policies

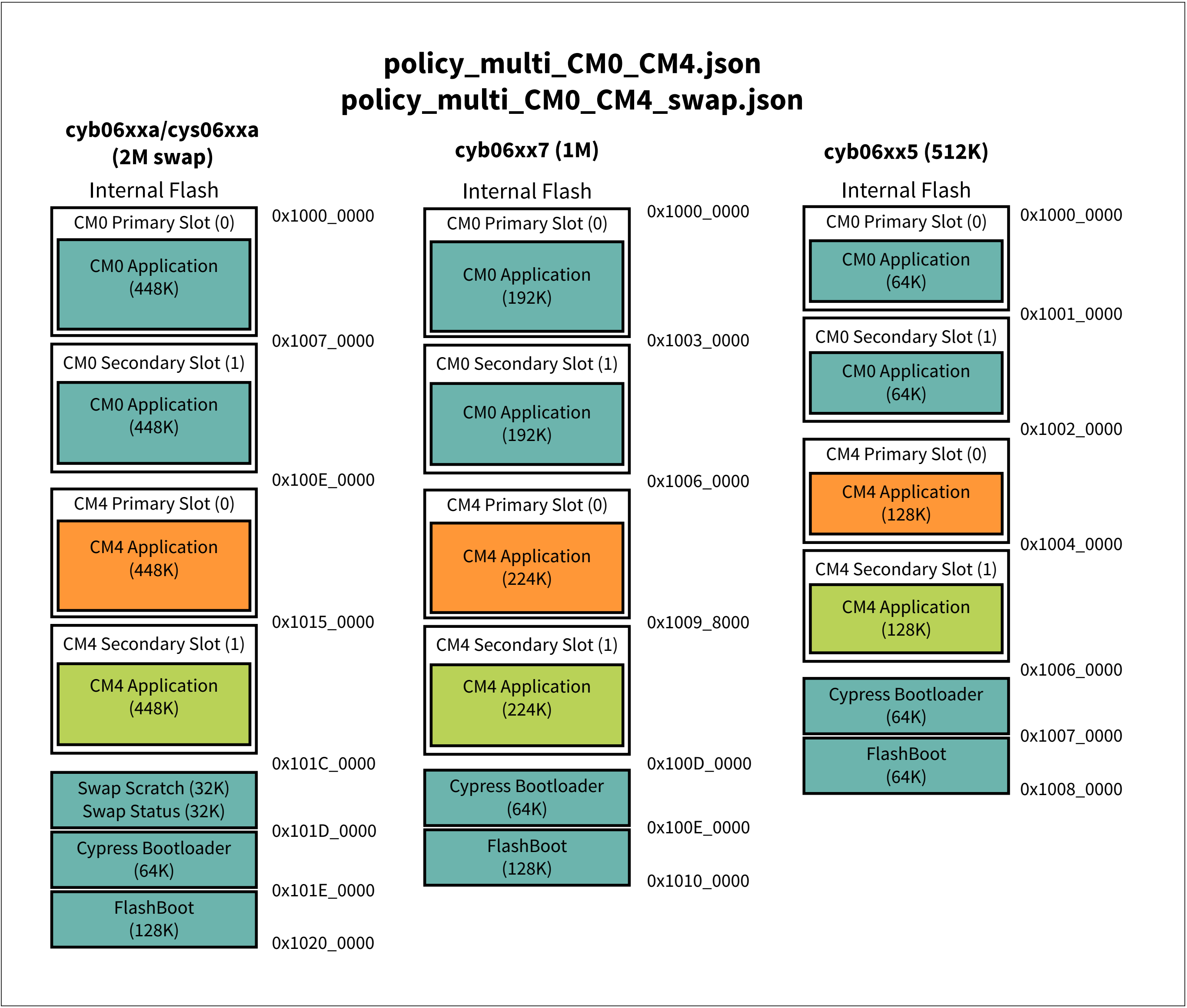

Figure 14 **Flash memory map for policy_multi_CM0_CM4 example policies**

6.3 Flash memory map for policy_single_CM0_CM4_smif example policies

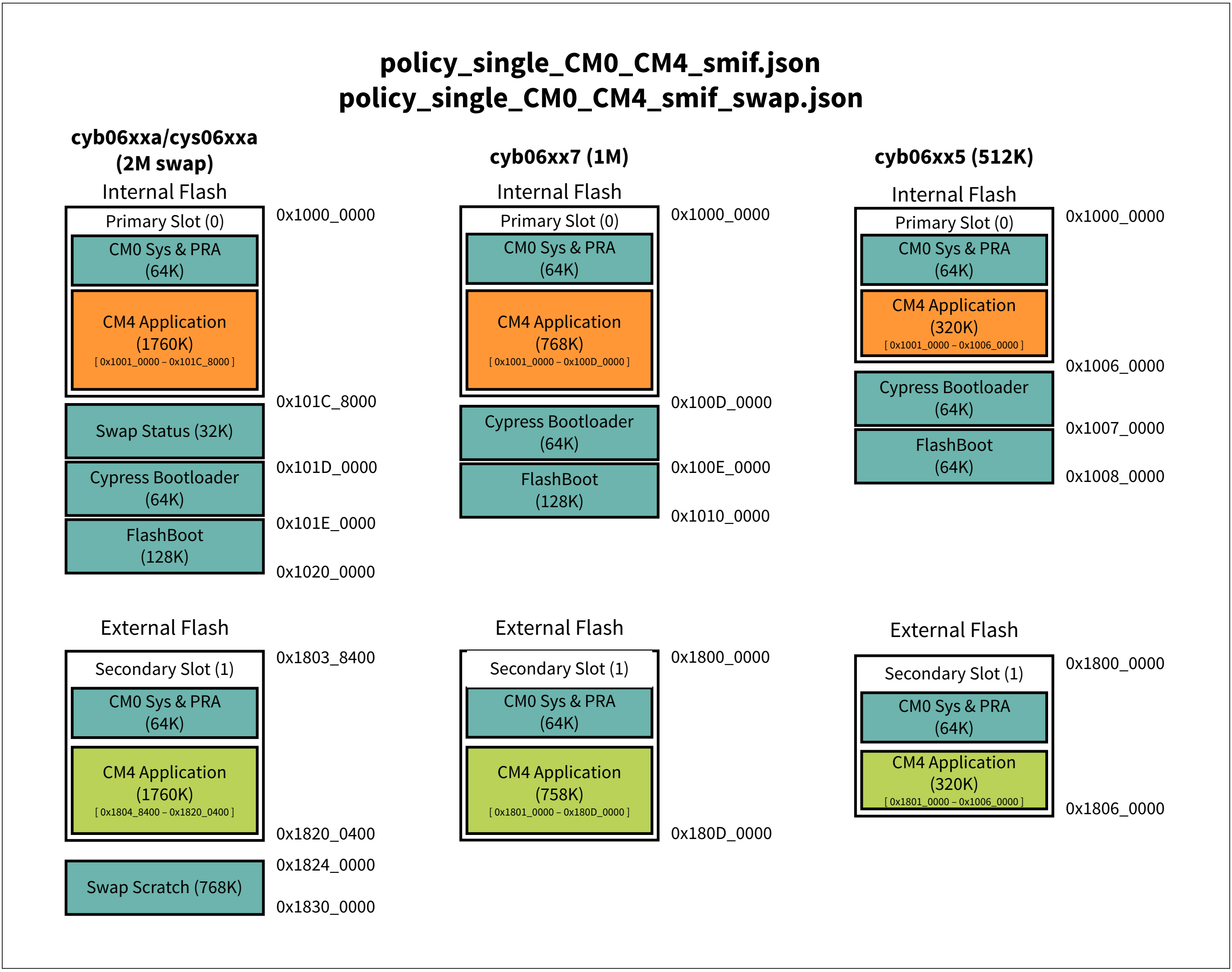

Figure 15 Flash memory map for policy_single_CM0_CM4_smif example policies

6.4 Flash memory map for policy_multi_CM0_CM4_smif example policies

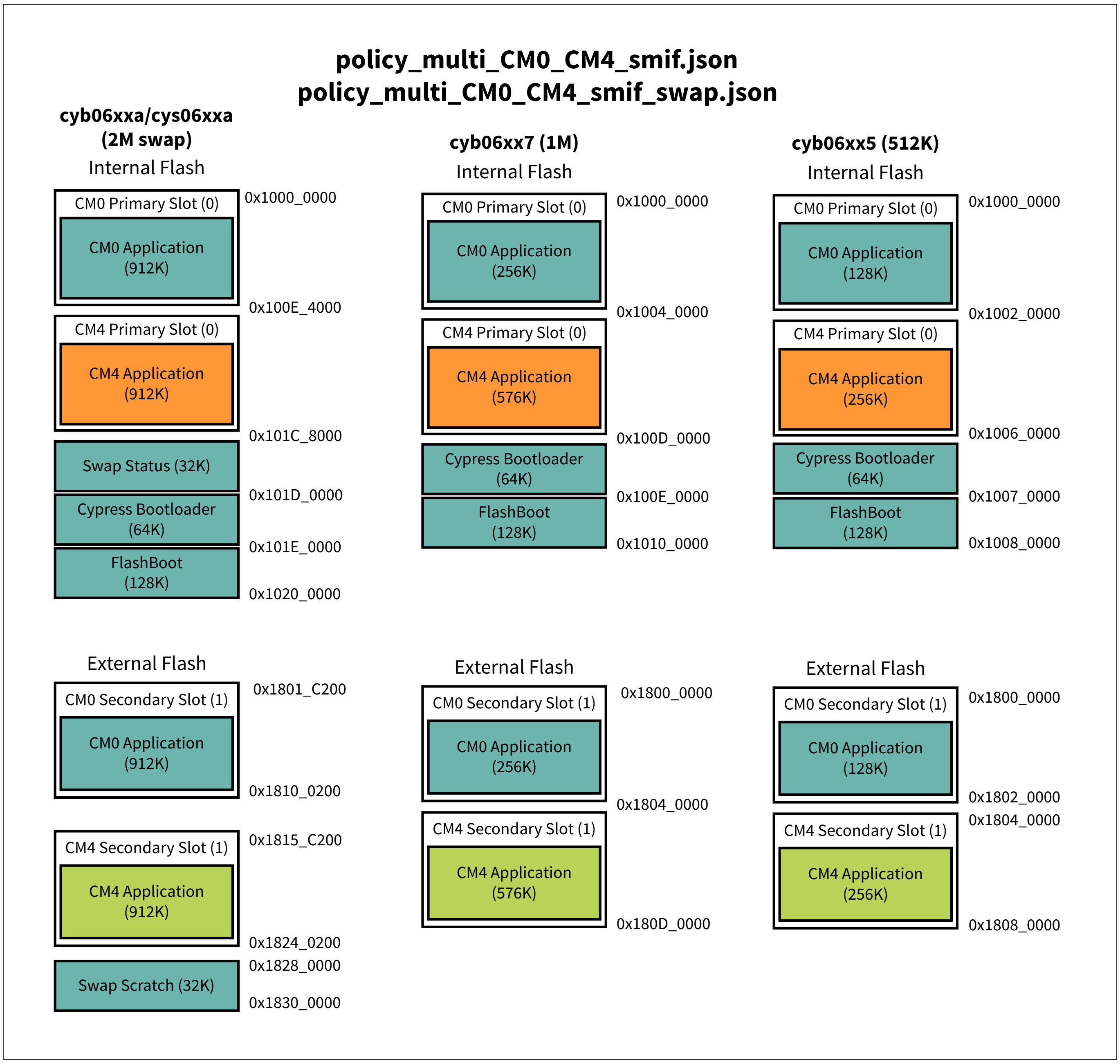

Figure 16 **Flash memory map for policy_multi_CM0_CM4_smif example policies**

Glossary

- **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 provides cryptographic processing. In the context of the PSOC™ 64 Secure Boot MCU, the HSM is a device programming engine located 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 injected keys, 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 values)
- **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 an RFC 7517 compliant data structure that represents a cryptographic key
- **Policies**: Policies are a collection of predefined (name, value) pairs that describe what is and is not allowed on the device. Most policies are enforced during boot time by the Root of Trust (RoT) firmware in the device, while some can be interpreted and enforced by higher layers of software, such as the Infineon Bootloader
- **Secure Boot**: Refers to a bootup process where the firmware 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® CPUs
- **CMSIS-DAP**: CMSIS-DAP is a specification and an implementation of 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 of application software running on Arm® Cortex® CPUs
- **KitProg3**: This is Infineon's low-level communication firmware for programming and debugging. It runs on a PSOC™ 5LP device. It is a multi-functional system that uses SWD for programming and debugging, and provides a USB-I^{2}C bridge and USB-UART bridge. It supports CMSIS-DAP and DAPLink
- **Single/Multi-image**: There are two 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 simultaneously, requiring a single update binary. In Multi-image mode, the CM0+ Secure co-processor and CM4 binaries are separate and can be updated individually with two different update binaries. The default is Single-image mode, as few customers need to modify the secure CM0+ Secure co-processor code base
- **SMIF**: Serial Memory Interface refers to the high-speed Quad-SPI interface on the PSOC™ 6 MCU in the context of this user guide
- **Rollback counter**: A special counter accessed by secure boot code holds the value of the latest valid image and is 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

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”.

Revision history

Document revision	Date	Description of changes
*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.
*H	2024-02-08	Updated section How to obtain the Secure Boot SDK and section CySecureTools installation and documentation.
*I	2024-09-13	Updated the Python version and Secure Policy Configurator section.
*J	2025-03-07	Template update, Revised content.

Trademarks

Edition 2025-03-07

Published by

Infineon Technologies AG

81726 Munich, Germany

Do you have a question about any aspect of this document?

Email: erratum@infineon.com

Document reference
IFX-uxa1726320808864