PSOC™ Control provisioning specification

About this document

Scope and purpose

This application note provides essential information for enabling the provisioning process of Infineon's PSOC™ Control devices. It details the RAM application, Edge Protect Tools, and policy information.

Intended audience

This document is intended for users with advanced knowledge of the various components involved in the PSOC™ Control provisioning process.

This document presumes that you are already familiar with AN238329 - Getting started with PSOC™ Control C3 MCU on ModusToolbox™ software.

Table of contents

About this document . . . 1
Table of contents . . . 2
1 Introduction . . . 3
1.1 Provisioning overview . . . 3
1.2 Re-provisioning . . . 4
2 Policy . . . 5
2.1 Debug policy . . . 6
2.2 Re-provisioning policy . . . 8
2.3 User keys . . . 9
3 Edge Protect Tools . . . 10
3.1 Overview . . . 10
3.2 Device communication . . . 11
3.3 Edge Protect Tools package for PSOC™ Control . . . 11
3.3.1 Packaged assets . . . 14
4 DLM (Downloadable Module) package . . . 15
4.1 RAM application (RAM_APP) . . . 15
4.2 DLM package . . . 15
4.3 Creating a DLM package . . . 17
4.4 Download and launch of the DLM package . . . 18
4.4.1 Launch flow: Debugger (SYS-AP) . . . 18
4.4.1.1 Sample OEM provisioning Tcl script . . . 21
4.4.2 Launch flow: OEM application . . . 25
4.4.3 Launch flow: BootROM DFU . . . 27
5 Appendix . . . 29
5.1 RAM application (DLM) status codes . . . 29
5.2 Edge Protect Tools supported commands . . . 31
6 References . . . 33
Glossary . . . 34
Revision history . . . 37
Trademarks . . . 38
Disclaimer . . . 39

1 Introduction

Provisioning is the process of injecting authorized keys and/or policy configuration into a device. Original equipment manufacturers (OEMs) use the provisioning process to load OEM Root of Trust (RoT) keys and OEM policies into the PSOC™ Control device. The provisioning of OEM keys, certificates, and policies transfers device ownership from Infineon to the OEM.

PSOC™ Control devices can be provisioned using Infineon's Edge Protect Tools package, which creates and downloads a signed provisioning packet into the device. This provisioning packet includes a RAM application capable of performing the provisioning, along with input parameters for the RAM application, such as the policy file and keys.

This document provides detailed information about the various types of certificates, policies, RAM application, and downloadable module (DLM) packages, all of which are included in or generated by Edge Protect Tools.

This document is not a programming specification. Refer to AN238998 - PSOC™ Control C3 MCU programming manual.

1.1 Provisioning overview

The provisioning process allows the OEM to configure boot and debug-related policy settings. Using the `OEM_ROT_KEY`, the OEM can perform further provisioning of the device to modify policies, as shown in **Step 1** of Figure 1. There are no restrictions on the number of times a device can be provisioned during the development lifecycle stage.

Provisioning is required to perform the following operations on the device:

- Install or replace keys
- Install or replace the OEM policies (Eg. boot configurations, and debug polices.)

Once the device transitions into the production lifecycle stage, it can neither be provisioned, nor be reverted to the development lifecycle stage. In the production lifecycle state (LCS), you only have a single opportunity to perform key revocation, which allows you to replace the existing OEM ROT keys with new ones. This operation is irreversible and can only be performed once.

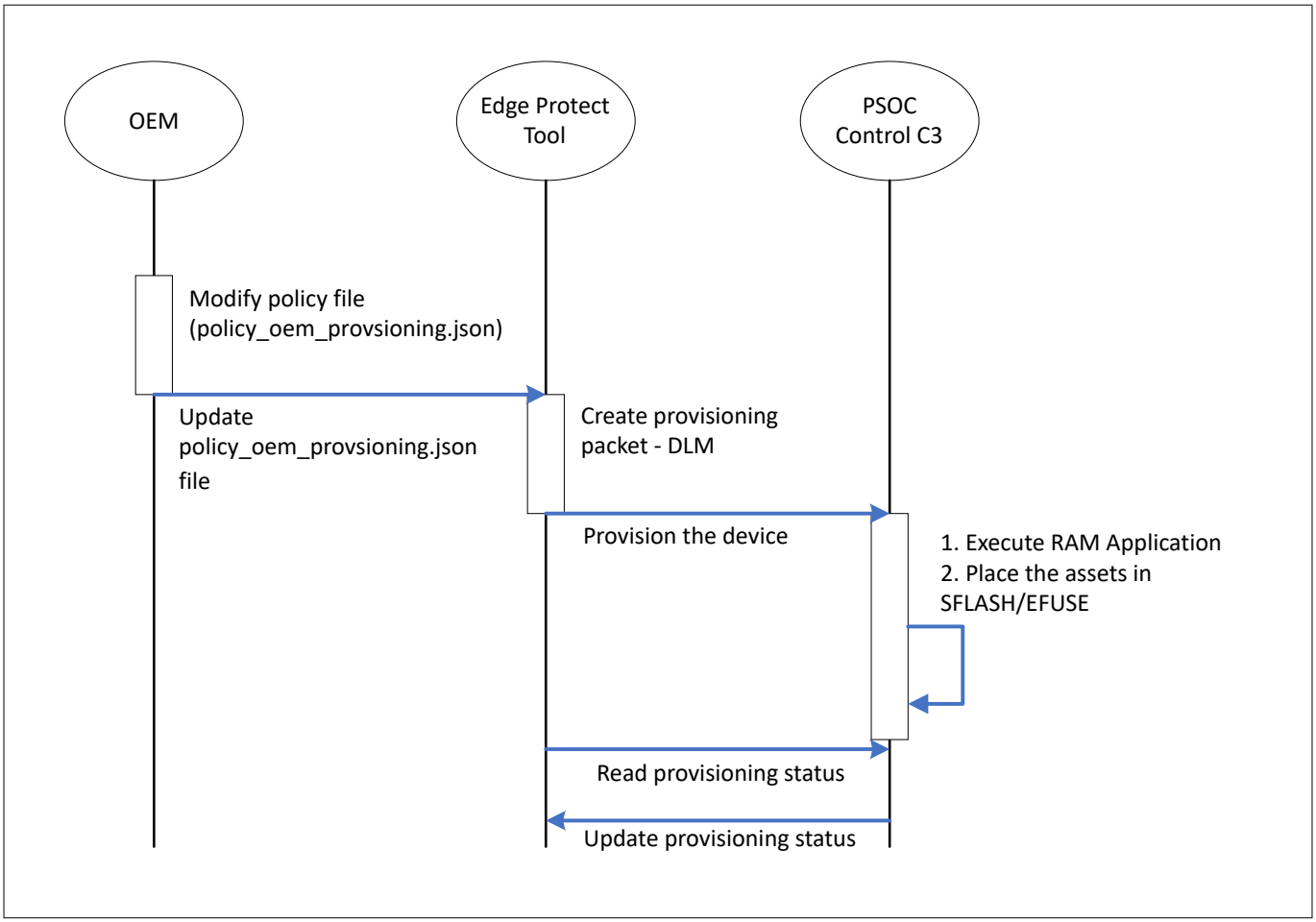


Figure 1 Provisioning overview

***Note**: To program an eFuse, VDDIO0 must be at 2.5 V ±10%. See the datasheet for more details.*

1.2 Re-provisioning

While in the production lifecycle stage, the device can only be re-provisioned if the re-provisioning policy for the specific policy field has been configured accordingly. For more details, see Re-provisioning policy

2 Policy

The boot configuration of PSOC™ Control devices is dictated by a policy file. All policy configurations are applied during device boot up by the Infineon BootROM (ROM_BOOT + FLASH_BOOT). The policy file is a JSON configuration file that defines the settings/configurations i.e., input parameters for device provisioning. Refer to AN240106 - Policy file for list of input parameters that can be provisioned to device.

In the Infineon factory, the device is provisioned with a set of default policy configurations. This default policy is packaged as a part of the Edge Protect Tools package and contains different sections that can be broadly classified into the three OEM policy sections:

- Debug policy
- Boot policy
- DFU policy
- MPC policy

The policy file also includes a section where the user OEM_ROT_KEY and device parameters are defined. The following figure shows what different input parameters are part of the policy file.

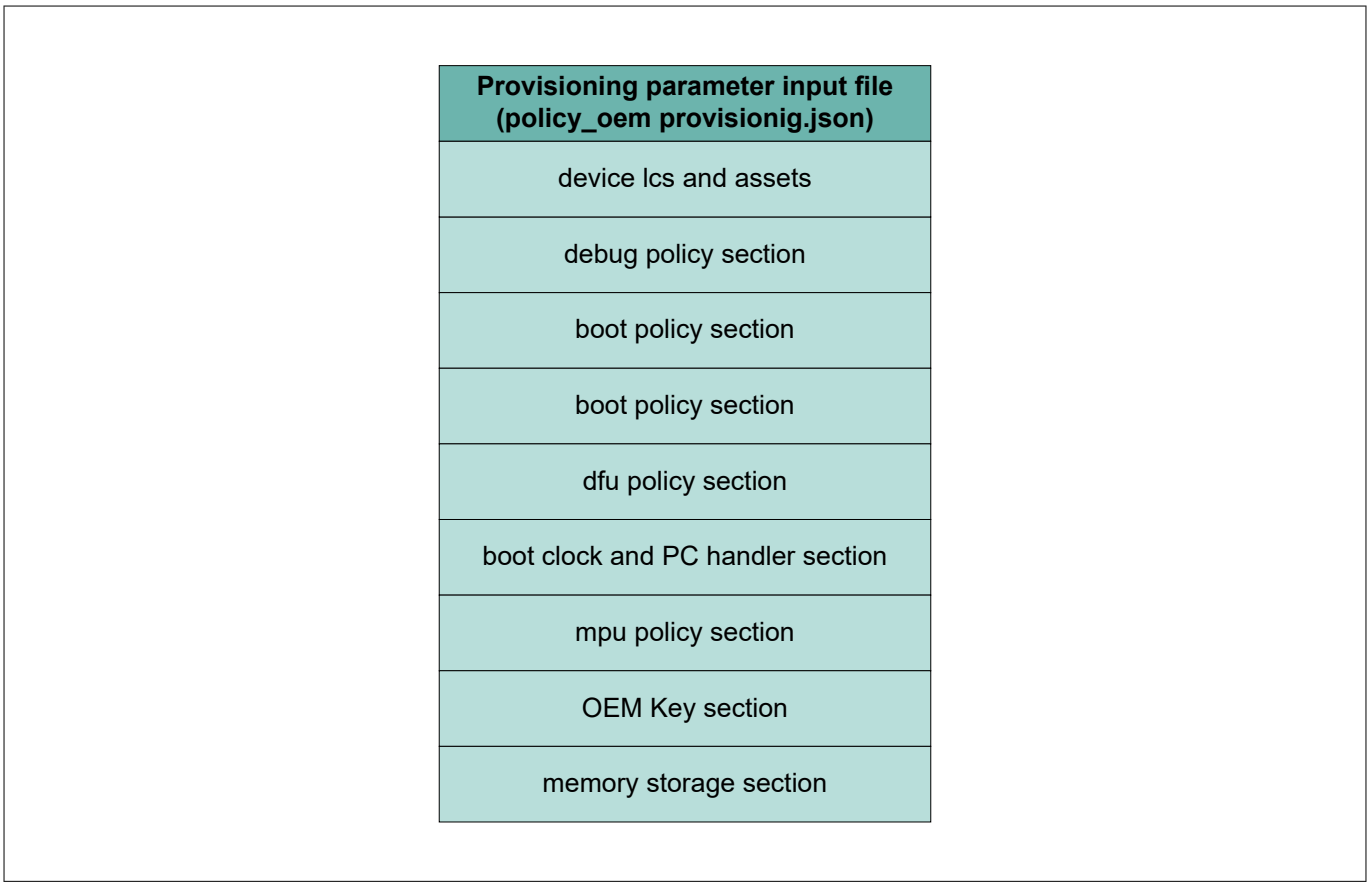


Figure 2 OEM policy file layout

Once the Edge Protect Tools package is initialized on the host PC, see the JSON policy in the `<application-directory>/policy/policy_oem_provisioning.json` file. This file can be edited using any text editor to customize the policy.

The policy file is converted into a CBOR packet that contains all provisioning data by Edge Protect Tools. This packet is then signed accordingly to the COSE format and the final packet is sent to the device during provisioning by Edge Protect Tools. The RAM application on the device then receives the packet, verifies the packet's signature, and processes the CBOR payload to get data for the provisioning. Typically, the RAM application processes provisioning data to convert it to the asset format stored on the device.

2.1 Debug policy

The debug policy configuration is used by the OEM to control access to debug ports through two mechanisms:

- Direct configuration
  - Enable/disable debug ports
  - Infineon's default policy enables all debug configurations
- Authenticated debug
  - Debug ports can be temporarily enabled using a debug token
  - Token must be cryptographically signed with authorized OEM RoT keys
  - Provides secure, controlled debug access when ports are disabled

***Note**: While Infineon's default policy enables all debug configurations, OEMs can implement authenticated debug for enhanced security by requiring valid debug tokens to access disabled ports.*

The authenticated debug works only when the debug field is set to “Disable” in the policy file. When set to “Disable”, the debug ports are permanently disabled and cannot be enabled even with debug token.

The default configurations in a policy file are shown in the following code block:

2 Policy

Debug default policy configuration

```
"debug": {
      "cpu": {
        "ap_cm33": {
          "description": "Configures the CM33 debug access port availability in the SECURE and
NORMAL_PROVISIONED LCS",
          "applicable_conf": "Enable, Disable, Permanently Disable",
          "value": "Enable"
        },
        "listen_window": {
          "description": "Configures the listen window time to acquire the CM33 debug access
port in the SECURE LCS",
          "applicable_conf": "100 ms, 20 ms, 1 ms, 0 ms",
          "value": "100 ms"
        },
        "cm33_dbg": {
          "description": "Configures the invasive debug of CM33 core",
          "applicable_conf": "Enable, Disable, Permanently Disable",
          "value": "Enable"
        },
        "cm33_nid": {
          "description": "Configures the non-invasive debug of CM33 core",
          "applicable_conf": "Enable, Disable, Permanently Disable",
          "value": "Enable"
        },
        "cm33_spid": {
          "description": "Configures the secure privileged invasive debug of CM33 core",
          "applicable_conf": "Enable, Disable, Permanently Disable",
          "value": "Enable"
        },
        "cm33_spnid": {
          "description": "Configures the secure privileged non-invasive debug of CM33 core",
          "applicable_conf": "Enable, Disable, Permanently Disable",
          "value": "Enable"
        },
        "cm33_secure": {
          "description": "Enables the CM33 secure AP interface",
          "applicable_conf": "Enable, Disable, Permanently Disable",
          "value": "Enable"
        }
      },
      "system": {
        "ap": {
          "description": "Configures the System debug access port availability in the SECURE
and NORMAL_PROVISIONED LCS",
          "applicable_conf": "Enable, Disable, Permanently Disable",
          "value": "Enable"
        },
        "secure": {
          "description": "Enables the system secure AP interface",
          "applicable_conf": "Enable, Disable, Permanently Disable",
          "value": "Enable"
```

```
        },
        "mmio_restrictions": {
          "description": "MMIO restrictions for System AP",
          "applicable_conf": "No restrictions, SRSS_GENERAL2 only, No access",
          "value": "No restrictions"
        },
        "debug_pins": {
          "description": "Configures debug pins and interface",
          "comment_1": "0000 - 2 pins (TMS, TCLK), JTAG is disabled",
          "comment_2": "0011 - 2 pins (TMS, TCLK), JTAG is enabled",
          "comment_3": "0101 - 4 pins (TMS, TCLK, TDO, TDI)",
          "comment_4": "0110 - 5 pins (TMS, TCLK, TDO, TDI, TRSTN)",
          "comment_5": "1001 - SWJ pins disabled, JTAG is enabled",
          "comment_6": "1010 - SWJ pins disabled, JTAG is disabled",
          "applicable_conf": "0000, 0011, 0101, 0110, 1001, 1010",
          "value": "0000"
        }
      }
    },
```

2.2 Re-provisioning policy

When the PSOC™ Control C3 device transitions to the production lifecycle stage, only once key revocation is permitted during the re-provisioning process.

If you re-provision the device, `OEM_ROT_KEY1` will be used to authenticate the OEM application instead of `OEM_ROT_KEY0`. Similarly, for the RAM application, `IFX_ROT_KEY1` will be utilized for authentication.

The default policy configurations are shown in the following code block:

Re-provisioning policy default configurations

```
{
  "policy": {
    "platform": "psoc_c3",
    "version": 1.0,
    "type": "reprovisioning"
  },
  "revoke": {
    "description": "Key revocation",
    "applicable_conf": "IFX_ROT_KEY_0, OEM_ROT_KEY_0, MPN_ROT_KEY_0, PROT_FW_ROT_KEY_0",
    "value": "OEM_ROT_KEY_0"
  },
  "public_key": {
    "description": "Path to the public key used to verify this policy signature",
    "value": "../keys/oem_rot_pub_key_0.pem"
  }
}
```

***Note**: Re-provisioning is not possible if the system-access port (SYS-AP) is disabled.*

2.3 User keys

The OEM can specify their own RoT keys, referred to as 'user keys,' in the OEM policy file. These keys are stored in secure flash memory during the provisioning process. These keys are used for signing the debug tokens used for authenticated debug.

The default policy configurations are shown in the following code block:

User keys default configurations

```
"pre_build": {
    "keys": {
      "oem_rot_public_key_0": {
        "description": "Path to the OEM RoT public key 0",
        "value": "../keys/oem_rot_pub_key_0.pem"
      },
      "oem_rot_public_key_1": {
        "description": "Path to the OEM RoT public key 1",
        "value": "../keys/oem_rot_pub_key_1.pem"
      }
    }
```

3 Edge Protect Tools

3.1 Overview

Edge Protect Tools is a Python-based package that provides a command-line interface for interacting with the MCU. It supports various functionalities, including provisioning, device integrity checks, RMA transition, and more. This tool is implemented in the Python programming language and uses several third-party Python modules to manage configuration files (`JSON`), object files (`ELF`), and perform cryptographic operations.

The tool can be accessed using any shell-based command-line interface. However, Infineon recommends using modus-shell, which is included in the ModusToolbox™ installation package, for seamless interaction with the Edge Protect Tools package.

When executing commands that require MCU interaction, such as provisioning, the tool prepares the necessary provisioning packets and communicates with the device using programming tools. It supports tools like OpenOCD via the SWD interface or the DFU Host Tool via serial interfaces such as I^2C, SPI, or UART.

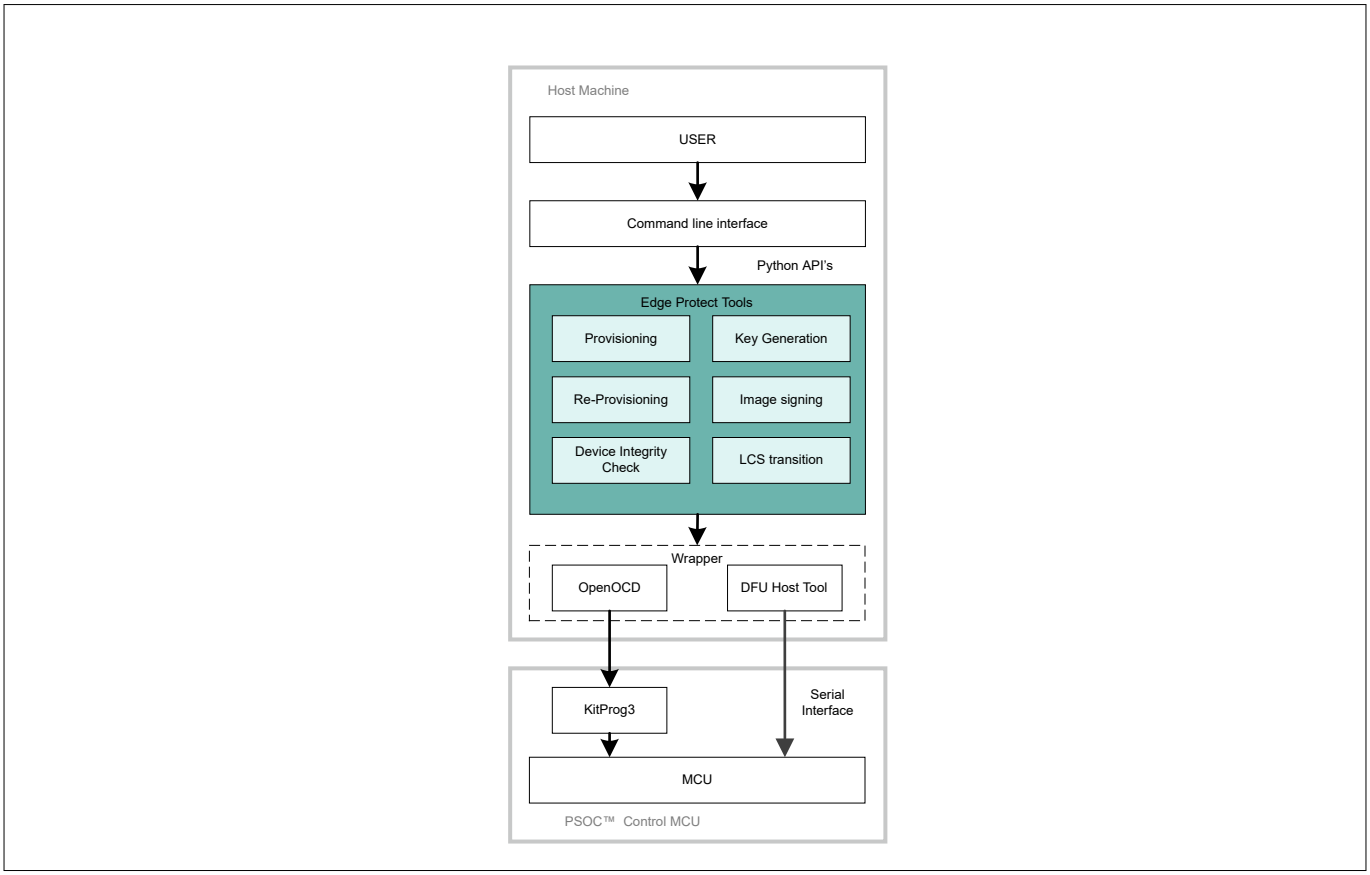


Figure 3 Edge Protect Tools overview

The operations supported by the tool include:

- **Provisioning**: Injecting keys and policies into the device
- **Re-provisioning**: Modifying specific configurations after provisioning
- **Key pair generation**: Creating key pairs for provisioning and image signing
- **Certificate generation**: Generating certificates for debugging and RMA transition
- **Signing images**: Signing images with the OEM key

For the complete list of commands supported by Edge Protect Tools package, see the Appendix section.

3.2 Device communication

The Edge Protect Tools package supports communication with the device through programming tools such as OpenOCD via the SWD interface or the DFU Host Tool via serial interfaces like I^2C, SPI, or UART. The following table mentions the different interface support for various MCUs.

Table 1 Device Communication Interface Support

Device	SWD	UART	I2C	SPI
PSOC™ Control C3	Yes	Yes	Yes	Yes

Instead of directly invoking programming tool commands, the Edge Protect Tools package uses a wrapper mechanism to interface with these tools. This design is extendable, enabling potential integration with other programming tools.

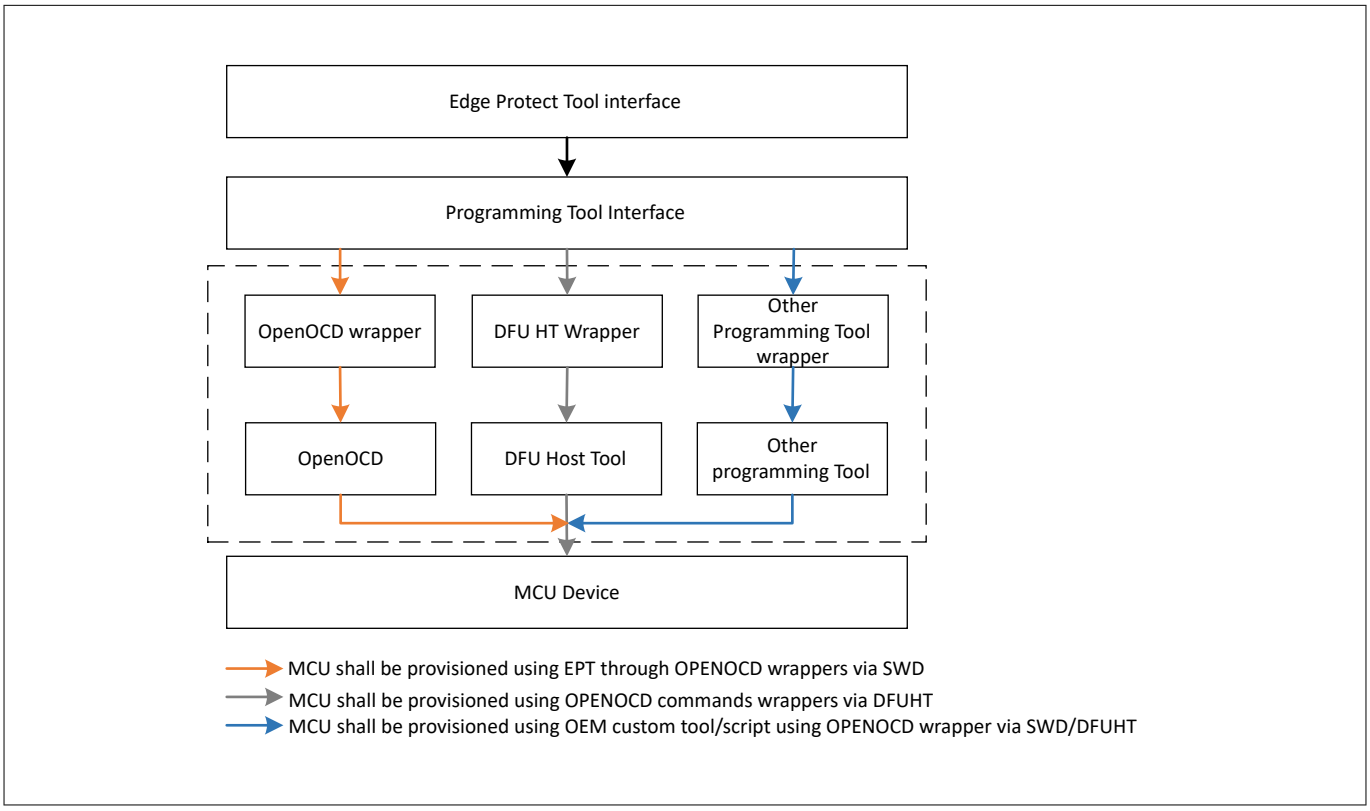


Figure 4 Edge Protect Tools device communication

3.3 Edge Protect Tools package for PSOC™ Control

Edge Protect Tools is included as part of the Edge Protect suite for the PSOC™ Control C3 device.

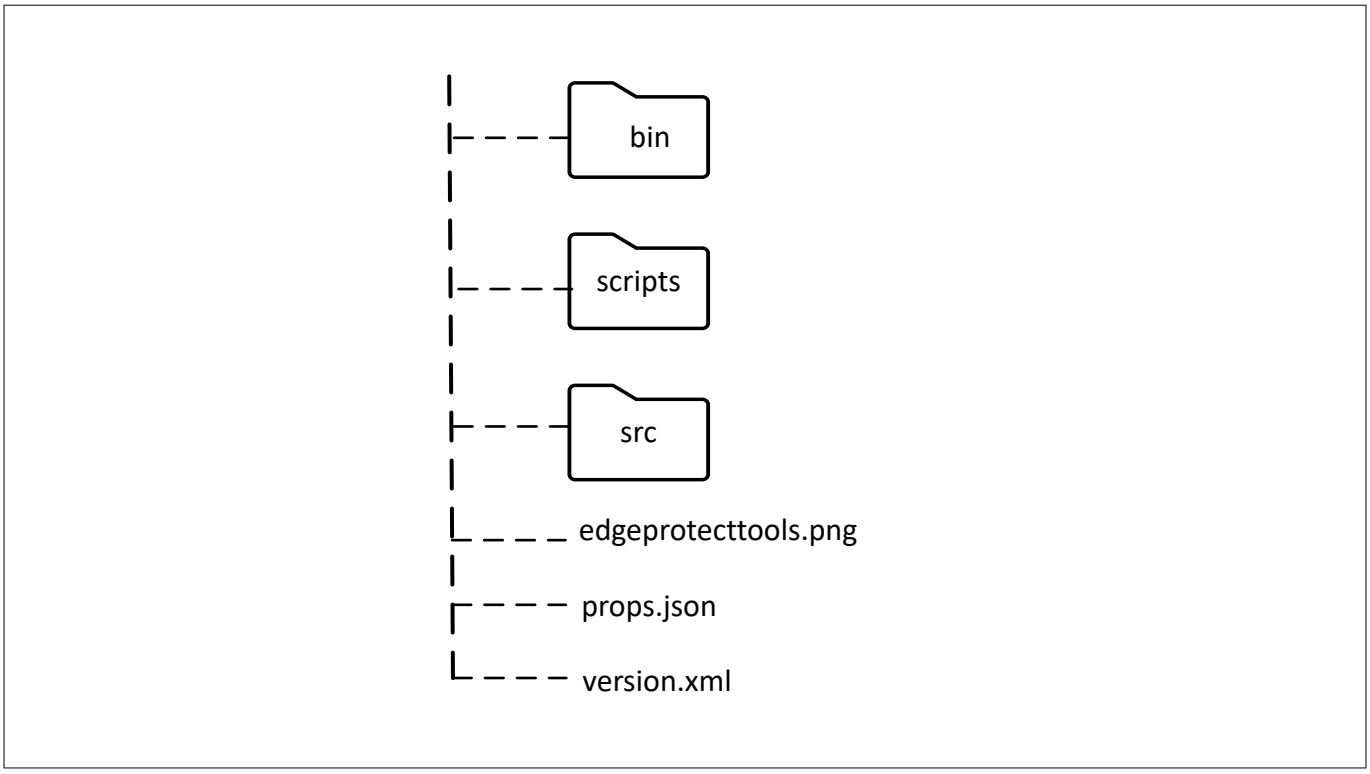


Figure 5 Edge Protect Tools in ModusToolbox™ Edge Protect Security Suite

The `./bin` directory contains a pre-compiled executable file for Windows, Linux, and macOS, which you can use directly. Add the executable path to your system's environment variable path:

```
<ModusToolbox-Edge-Protect-Security-Suite-install- path>/tools/edgeprotecttools/bin
```

Before starting, Edge Protect Tools must be initialized for a specific target, such as the 'psoc_c3' for the PSOC™ Control C3.

To initialize, navigate to your working directory and execute the following command in a shell CLI:

```
edgeprotecttools -t psoc_c3 init
```

After initialization, the following folder structure as shown Figure 6 in will be created in your working directory.

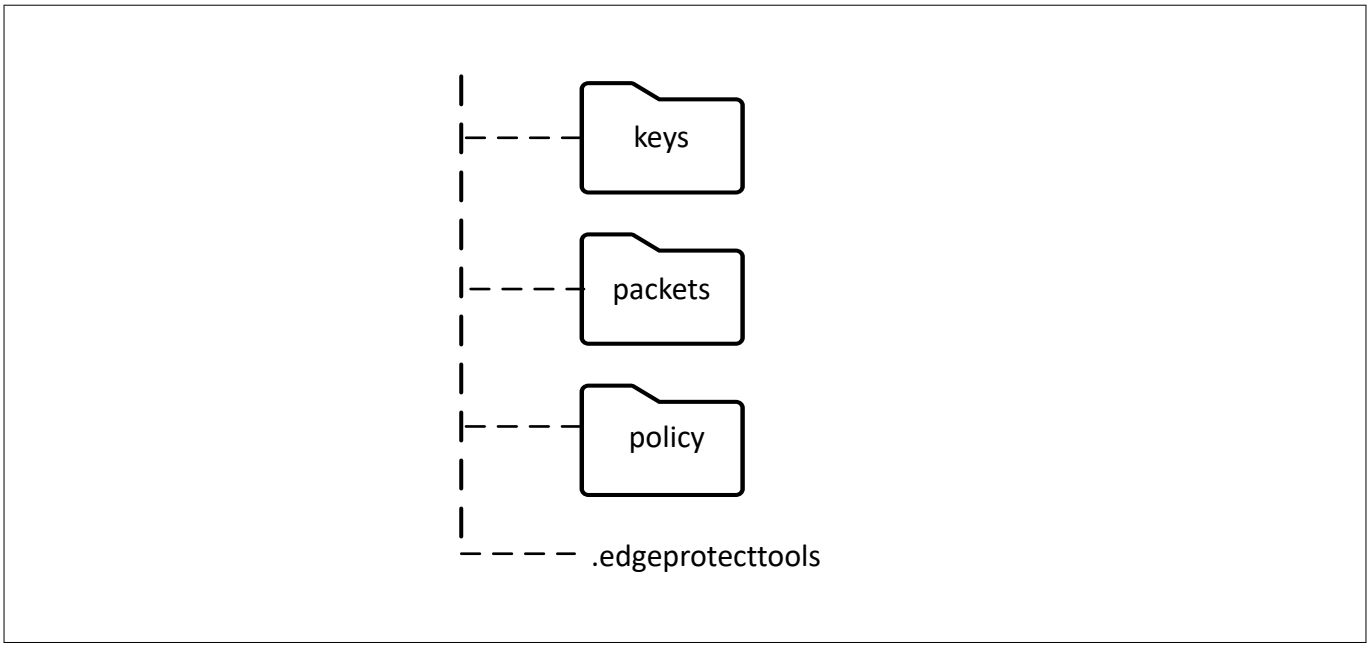


Figure 6 Edge Protect Tools project directory

Setting up the OpenOCD path:

To set up the OpenOCD path for use with the tools, execute the following command:

```
edgeprotecttools set-ocd --name openocd --path <openocd_path>
```

***Note**: Here, <openocd_path> is the path to the OpenOCD tool on your system. Typically, it will be located in the default installation directory: C:/Infineon/Tools/ModusToolboxProgtools-1.3/openocd.*

Installing Edge Protect Tools from the source:

If you need to make changes to the source files, you must install the tools from the source after editing. Follow the steps below to install Edge Protect Tools from the source.

- On Windows, ensure that the following Python installation paths are added to the `Path` environment variable. Reorder them so they appear at the top of the list:

  ```
  <python-install-path>/python
  <python-install-path>/python/Scripts
  ```

- If you are using Linux or macOS, grant superuser privileges to the source directory by running the following command before installation:

  ```
  sudo chmod 0777 <ModusToolbox-Edge-Protect-Security-Suite-install- path>/tools/
  edgeprotecttools/src
  ```

- On windows, open modus-shell and execute the following command. For Linux or macOS, run the same command in their respective terminals:

  ```
  pip install --upgrade --force-reinstall <ModusToolbox-Edge-Protect-Security-Suite-install-
  path>/tools/edgeprotecttools/src
  ```

***Notes**:*

1. *On Windows, use a Windows-style path (that is, not a path like `/cygdrive/c/`). Additionally, use forward slashes (/). For example, `C:/Users/XYZ/ModusToolbox/`*
2. *After installing Edge Protect Tools on Linux or macOS, follow any additional instructions provided by `pip` during the installation process*

3.3.1 Packaged assets

The following assets are included in the Edge Protect Tools package and can be seen in the `<working-dir>` where the Edge Protect Tools is initialised

- **OEM policy**: The OEM policy file (`policy_oem_provisioning.json`) is used to configure device policy settings. This file is located in the `<working-dir>/policy` folder
- **RAM application**: Infineon-signed RAM application (DLM) binaries are available in the `<working-dir>/packets/apps` folder. RAM applications are owned by Infineon and cannot be customized by users
- **JSON template files**: JSON template files used by Edge Protect Tools are located in the `<working-dir>/packets` folder

4 DLM (Downloadable Module) package

A DLM package is a structured binary file that contains:

- Infineon signed RAM_APP
- Input parameters for device provisioning
- Signatures

4.1 RAM application (RAM_APP)

RAM applications are executable code loaded into the SRAM and executed in the PC0 (Protection Context 0) (AN240106 – Getting started with PSOC™ Control C3 security). Certain device features, such as provisioning and re-provisioning, are implemented using RAM applications or DLMs. These applications are developed by Infineon, signed with Infineon keys, and verified and launched by the device's BootROM. It is important to note that RAM applications cannot be developed by OEMs.

RAM applications are packaged in the MCUboot image format. This format adds application metadata to verify image integrity and authenticity, along with custom metadata in the tag-length-value (TLV) format.

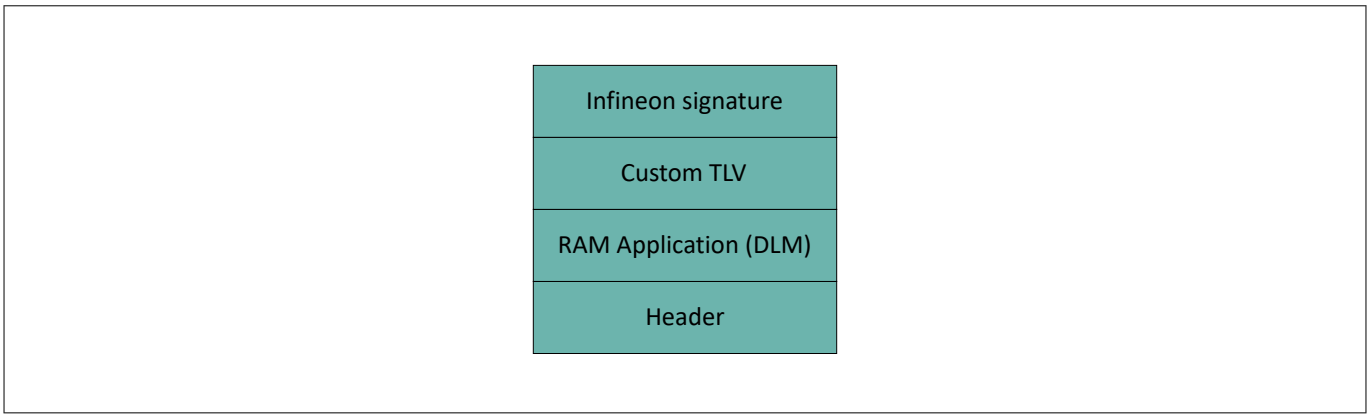


Figure 7 Signed RAM application

4.2 DLM package

As mentioned, DLM package consists of the following components:

- Infineon signed RAM application
- DLM input parameters

The DLM packages are signed only for a few policies such as reprovisioning, RMA and DFU. For default device provisioning, the DLM signing is not required. The table below mentions which policies require signing.

Table 2 Policies requiring signatures

App name	Input parameters signed		DLM package signed	
	NORMAL_PROVISIONED LCS	SECURE LCS	NORMAL_PROVISIONED LCS	SECURE LCS
cyapp_prov_oem	NA	NA	NA	NA
cyapp_reprovisioning	Yes	Yes	Yes	Yes
cyapp_to_rma	NA	Yes	NA	Yes
cyapp_prot_fw_dfu	NA	NA	NA	Yes
cyapp_prot_fw_policy	Na	NA	NA	NA

***Note**: In the above table, the 'Yes' fields required OEM_ROT_KEYS for signing.*

To sign the DLM package, it is wrapped in the MCUboot format with a header and signature. The package is signed using the `OEM_ROT_PRIVATE_KEY`. The ROM boot always expects the DLM package to follow this format for processing; however, the package signature is verified only in the production lifecycle stage (SECURE LCS).

The OEM signature provides the OEM with control over which DLM is executed on the device in the production LCS, as well as its associated input parameters. To modify the input parameters within the package, the package must be re-signed using the `OEM_ROT_PRIVATE_KEY`.

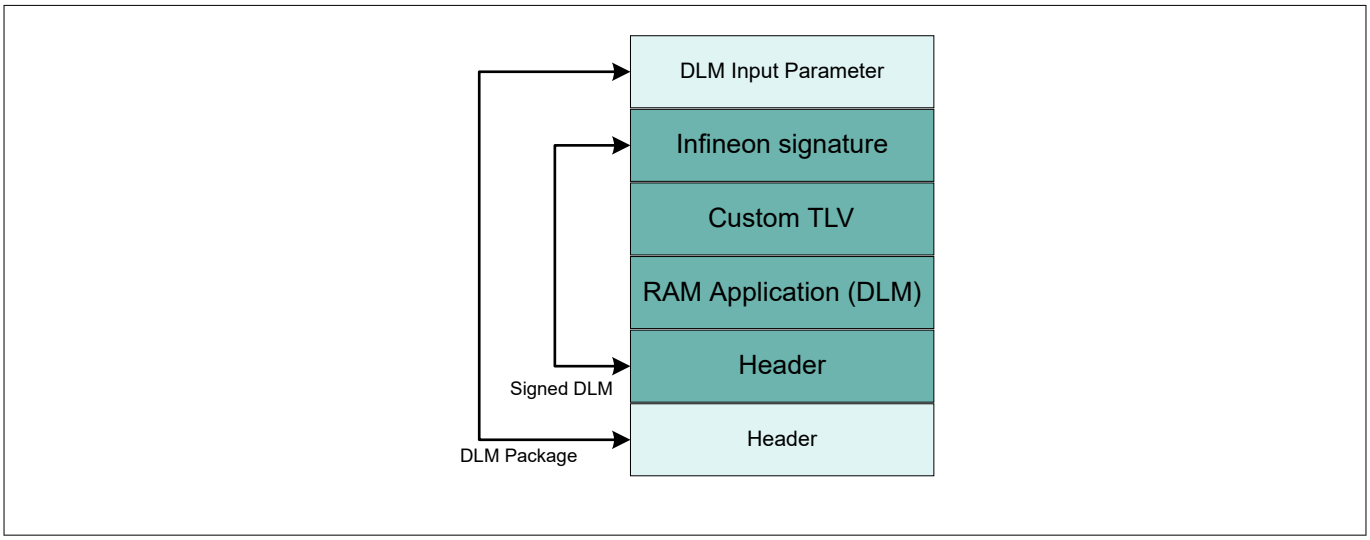


Figure 8 DLM package format

The placement of the DLM package in the device's SRAM memory is demonstrated in Figure 9. The DLM package is uploaded into the internal SRAM using either the debugger (SYS-AP) or an application firmware running on the device. Upon request, the BootROM validates the DLM package and post a successful validation, it launches the RAM application in SRAM.

***Note**: In case of OEM provisioning, the DLM need not be signed and the DLM package will not contain the OEM signature.*

Once launched, the RAM application reads the input parameters from the device's SRAM and provisions them accordingly in the SFlash region.

The SRSS registers (BOOT_DLM_CTL) handle

- DLM package validation requests to BootROM
- RAM application launch control
- RAM application execution status reporting

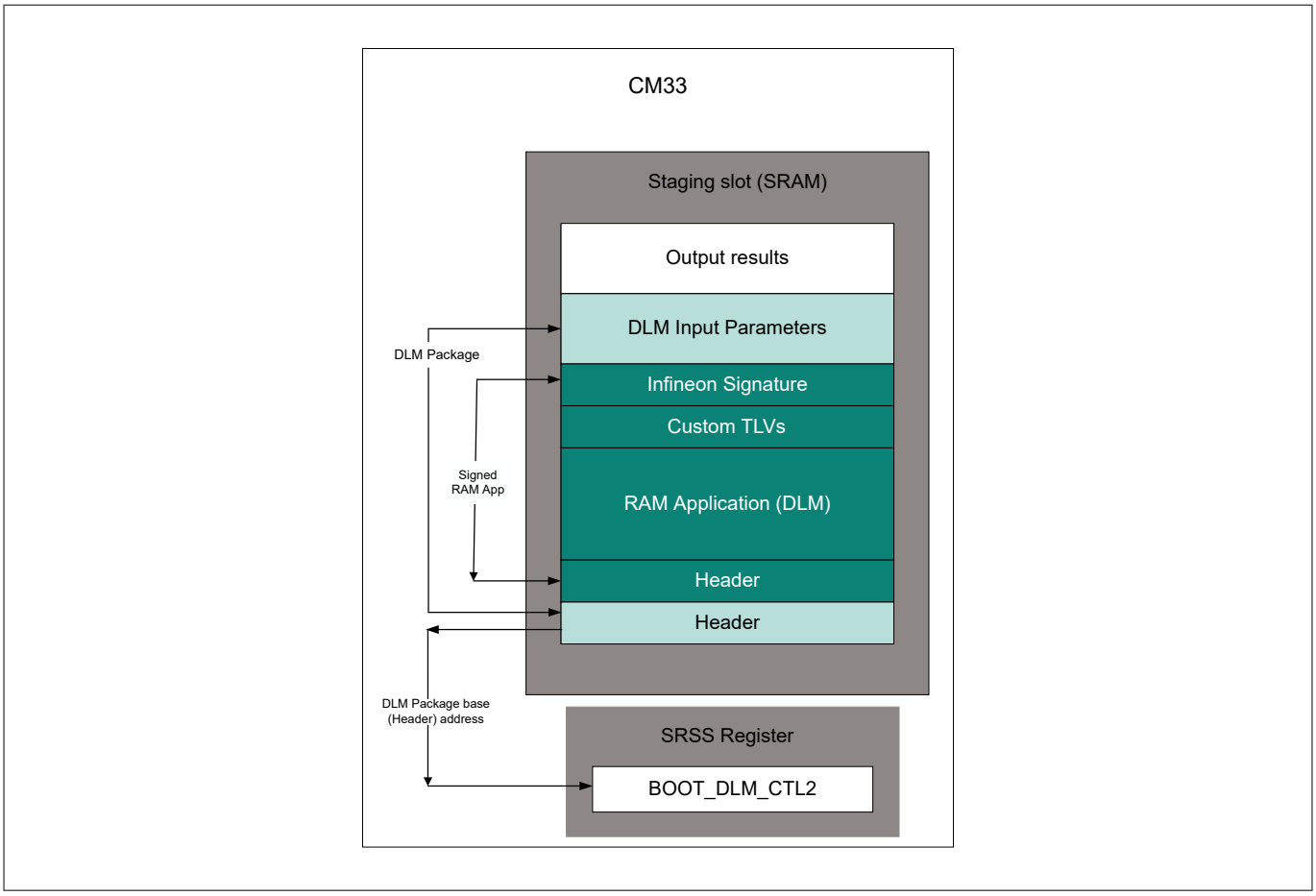


Figure 9 **DLM package storage**

Table 3 **SRSS registers**

SRSS register	Description
BOOT_DLM_CTL	Post requests for DLM package validation to the BootROM
BOOT_DLM_STATUS	Report the RAM application execution status in SRAM
BOOT_DLM_CTL_2	Store the DLM package base address in SRAM required by BootROM for validation
RES_SOFT_CTL	Trigger a device reset using the debugger (SYS-AP)

The input parameters, which are part of the DLM package placed in the device's SRAM, have their base address and maximum size stored by the ROM boot at a fixed SRAM location. This information is used by the RAM application to access the input parameters.

Similarly, the output results populated by the RAM application in the device's SRAM have their output result buffer base address and maximum size stored by the ROM boot at a fixed SRAM location. This information is used by the RAM application to write the output results.

See the Appendix section for the list of RAM application status codes.

4.3 Creating a DLM package

Edge Protect Tools generates the DLM package from the pre-signed Infineon RAM application using the DLM parameters provided as input. The resulting DLM package includes the OEM signature, which is created using the OEM_ROT_PRIVATE_KEY. Additionally, the pre-signed Infineon RAM application binaries are included as

part of the Edge Protect Tools package. Refer to the PSOC™ Control C3 MCU: OEM Provisioning code example for DLM package creation for a RAM Application.

DLM package is created when *edgeprotecttools* init command is run. It can be found in `<Workspace>/<CodeExampleName>/packets/apps/prov_oem/prov_oem_dlm.hex`

The DLM package can also be explicitly created with command below after running the edgeprotecttools init command.

```
edgeprotecttools -t psoc_c3 build-ramapp-package --app ./packets/apps/prov_oem/
cyapp_prov_oem_signed_ifx0.bin --inparams ./packets/apps/prov_oem/in_params.bin --hex-addr
0x32008000 --output ./packets/apps/prov_oem/prov_oem_dlm.hex
```

cyapp_prov_oem_signed_ifx0.bin - Infineon signed RAM_APP

in_params.bin - Input parameters to be provisioned to device

prov_oem_dlm.hex - Generated DLM hex file

0x32008000 - SRAM location where the DLM is programmed

4.4 Download and launch of the DLM package

This section provides details on downloading and launching the RAM application using different methods available for the device.

Method #1 uses an external debugger, which requires a physical connection to the device through the serial wire debug (SWD) interface.

Method #2 uses OEM application to download the programmed DLM from flash to SRAM and launch.

Method #3 uses BootROM DFU to download the DLM package to device.

4.4.1 Launch flow: Debugger (SYS-AP)

This launch flow defines the steps required to program a DLM package, verified, and launched by the BootROM to provision the device. As a prerequisite for this flow, ensure the system access port (SYS-AP) remains enabled, as disabling it will result in permanent loss of device access. See the Debug policy section for more details. Along with the flow, the OpenOCD implementation is also mentioned. You can find the TCL script in Sample OEM provisioning Tcl script section, which implements this launch flow to provision the device in OpenOCD.

Flow details:

1. Establish communication with the device. Example:SWD

```
source [find interface/kitprog3.cfg]
```

2. Read the lifecycle (LCS) state from the device

```
set bootrow [expr {[mrw $EFUSE_BOOTROW] & 0xFFFF}]
```

3. Check the BootROM status and trigger a soft reset

```
if {(([mrw $BOOT_DLM_CTL] & $TST_DBG_CTL_WFA_Msk) == 0)} {
                    mww $BOOT_DLM_CTL$TST_DBG_CTL_WAIT_RAM_APP_RQST
                    }
```

4. Verify that the BootROM is waiting for the RAM application

```
(([mrw $BOOT_DLM_CTL] & $TST_DBG_CTL_WFA_Msk) == 0
```

5. Prepare and load the RAM application and input parameters as a DLM package

```
"$path_tools" -t psoc_c3 create-provisioning-packet -p policy/policy_oem_provisioning.json
                    "$path_tools" -t psoc_c3 build-ramapp-package --app packets/apps/
prov_oem/cyapp_prov_oem_signed_ifx0.bin --inparams packets/apps/prov_oem/in_params.bin --
output packets/apps/prov_oem/prov_oem_dlm.hex
                    load_image $image_file
```

***Note**: The cyapp_prov_oem_signed_ifx0.bin is the signed RAM application, in_params.bin is the input parameters. Both these binaries are combined into a single DLM package named prov_oem_dlm.hex which is loaded to the device SRAM*

6. Notify the BootROM that the DLM package is uploaded and is ready for validation

```
mww $BOOT_DLM_CTL2 $IMAGE_SLOT_START_ADDR
                    mww $BOOT_DLM_CTL 0
```

7. Wait for application completion if launched by the BootROM
8. Verify that the application is executed successfully by checking the boot status register (0xF2A00001)

```
mdw $BOOT_DLM_STATUS
```

4 DLM (Downloadable Module) package

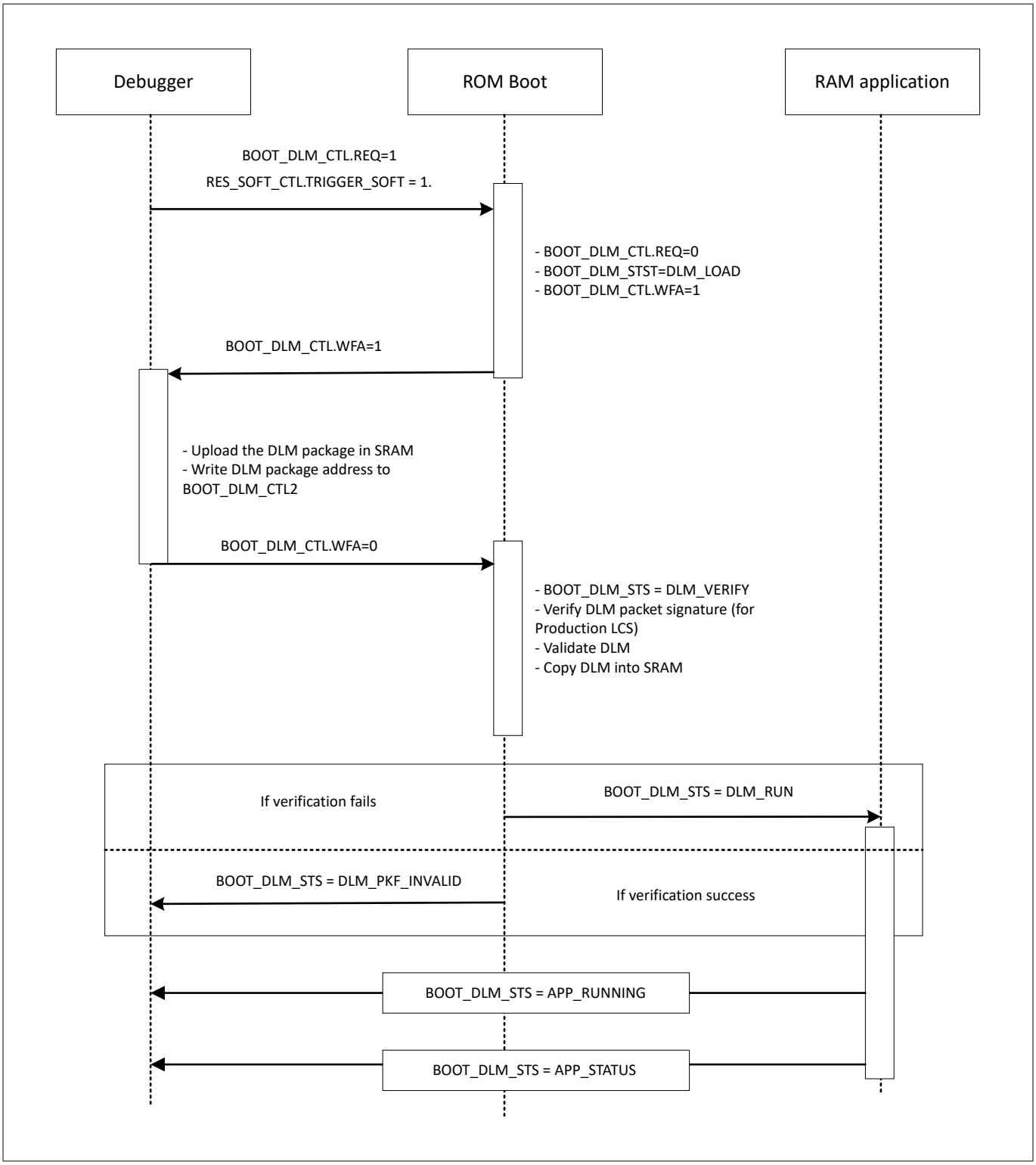


Figure 10 Launch DLM by debugger flow

4.4.1.1 Sample OEM provisioning Tcl script

Following is an example Tcl script designed to provision the device using the OpenOCD command. This script follows the steps outlined in the previous section. To use the script, execute it within the OpenOCD environment.

```
openocd.exe -s ./scripts -f <example_script_file.tcl>
```

```
#/c/Infineon/Tools/ModusToolboxProgtools-1.3/openocd

# Default parameters values
dict set app_data image_path prov_oem_dlm.hex
dict set app_data in_params_addr 0x3400D000
dict set app_data in_data {}
dict set app_data out_params_addr 0x3400E000
dict set app_data out_data {}
dict set app_data out_data_size 0
# A timeout in 100 ms units, e.g. 100 means 100 * 100ms = 10_000 ms = 10 sec
dict set app_data wait_for_app_timeout 200
dict set app_data reset_after_app_cmplt 0

# Check if we have specific folder as argument
if {0 == [info exists app_folder]} {
    set app_folder "."
    puts "\nUsing current folder \"[pwd]\""
} else {
    puts "\nUsing app folder \"$app_folder\""
}

# Check if we have specific config as argument
if {0 == [info exists config_file]} {
    set config_file config.conf
    puts "\nUsing default config file \"$config_file\""
} else {
    puts "\nUsing config file \"$config_file\""
}

set config_file $app_folder/$config_file

# Parsing config file parameters and overwrite app_data
foreach line [split [read [open $config_file]] \n] {
    if {[string match "#*" $line]} {
        # This is commented string
    } else {
        if {[regexp {^([^=]*)=(.*)$} $line -> key value]} {
            dict set app_data $key $value
        } else {
            # "=" absent in the line
        }
    }
}
```

```
# Parameters that could be changed using config file
set image_file             $app_folder/[dict get $app_data image_path]
set in_data_file           [dict get $app_data in_data]
set out_data_file          [dict get $app_data out_data]
set out_data_size          [dict get $app_data out_data_size]
set in_data_addr           [dict get $app_data in_params_addr]
set out_data_addr          [dict get $app_data out_params_addr]
set wait_for_app_timeout   [dict get $app_data wait_for_app_timeout]
set reset_after_app_cmplt  [dict get $app_data reset_after_app_cmplt]

# Address to put application signed image in MCUBoot format
set IMAGE_SLOT_START_ADDR  0x34003000

# Hardware registers
set BOOT_DLM_CTL     0x52200404
set BOOT_DLM_CTL2    0x52200408
set BOOT_DLM_STATUS  0x5220040C
set RES_SOFT_CTL     0x52200410
set BOOT_STATUS      0x52200418
set EFUSE_BOOTROW    0x52610180

# Registers constants
set RES_SOFT_CTL_RESET_RQST 0x00000001
set TST_DBG_CTL_WAIT_RAM_APP_RQST 0x00000001
set TST_DBG_CTL_WFA_Msk 0x80000000

# Intermidiate states returned using BOOT_DLM_STATUS
set CYBOOT_WFA_POLLING              0x0D500080
set CYBOOT_SERVICE_APP_LAUNCHED     0x0D500081
set CYBOOT_SERVICE_APP_NOT_LAUNCHED 0x0D500082
set CYAPP_APP_RUNNING               0xF2A00010

set EXIT_DUE_ERROR 1
set DEFAULT_TIME_OUT_MS 100

# Interface settings
# Uncomment when power supply enabling not needed
# set ENABLE_POWER_SUPPLY 0
set ENABLE_ACQUIRE 0

# Only one interface should be selected (uncommented). Other left for reference.
# source [find interface/jlink.cfg]
source [find interface/kitprog3.cfg]
# source [find interface/cmsis-dap.cfg]

transport select swd
source [find target/infineon/psc3.cfg]
psc3.cm33 configure -defer-examine
adapter speed 100
init
#targets $_CHIPNAME.sys
targets psc3.sys
```

```
puts "\n"

set bootrow [expr {[mrw $EFUSE_BOOTROW] & 0xFFFF}]
set is_virgin_lcs  [expr {($bootrow == 0) ? 1 : 0}]
set is_sort_lcs    [expr {($bootrow == 0x0029) ? 1 : 0}]
set is_rma_lcs     [expr {(($bootrow & 0x3000) == 0x3000) ? 1 : 0}]
set has_slow_clock [expr {$is_virgin_lcs || $is_sort_lcs || $is_rma_lcs}]

# Timeout [seconds] = 14e+6 [#clock for RSA signature] / CLK_FREQ [Hz]
# For IMO, CLK_FREQ=8 MHz
# For IHO, CLK_FREQ=50 MHz
# Add some margin (10-20%) to a timeout
set wfa_img_verify_timeout  [expr {$has_slow_clock ? 2000 : 250}]

# VIRGIN & RMA LCS: the WFA bit is set, the BootROM is waiting for application.
# Thus, no reason to reset a device.
# For other LCS, post a request and reset a device then wait for TST_DBG_CTL_WFA_Msk.
if {(([mrw $BOOT_DLM_CTL] & $TST_DBG_CTL_WFA_Msk) == 0)} {

    # Post request to BootROM to stop and wait for application after reset
    puts "Write BOOT_DLM_CTL.REQUEST = 0x001 and reset the device using
RES_SOFT_CTL.TRIGGER_SOFT = 1"
    mww $BOOT_DLM_CTL $TST_DBG_CTL_WAIT_RAM_APP_RQST

    # Reset device using $RES_SOFT_CTL register.
    # The mww command fails so catch this fail and continue
    local_echo off
    catch { mww $RES_SOFT_CTL $RES_SOFT_CTL_RESET_RQST }
    local_echo on

    puts "Waiting until BootROM stopped and ready for application upload..."
}

# Check if BootROM waits for application
set timeout 0
while {(([mrw $BOOT_DLM_CTL] & $TST_DBG_CTL_WFA_Msk) == 0)} {
    incr timeout 1
    if {$timeout > $wait_for_app_timeout} {
        puts "FAIL: BootROM did not set flag waiting for application - timeout"
        exit $EXIT_DUE_ERROR
    }
    sleep $DEFAULT_TIME_OUT_MS
}

if {([mrw $BOOT_DLM_STATUS] != $CYBOOT_WFA_POLLING)} {
    puts "FAIL: BootROM did not set expected BOOT_DLM_STATUS"
    puts [format "    BOOT_DLM_CTL: 0x%08X;"    [mrw $BOOT_DLM_CTL]]
    puts [format "    BOOT_DLM_CTL2: 0x%08X;"   [mrw $BOOT_DLM_CTL2]]
    puts [format "    BOOT_DLM_STATUS: 0x%08X;" [mrw $BOOT_DLM_STATUS]]
    puts [format "    BOOT_STATUS: 0x%08X;"     [mrw $BOOT_STATUS]]
    exit 1
} else {
    puts "Ready for application programming!"
```

```
}

# Set random value to check if BOOT_DLM_STATUS changed
mww $BOOT_DLM_STATUS 0x11223344

# Loading application
puts "Programming application: $image_file to $IMAGE_SLOT_START_ADDR"
#load_image $image_file $IMAGE_SLOT_START_ADDR

#the image should be in DLM format (RAM App _ input parameters)
#"$path_tools" -t psoc_c3 create-provisioning-packet -p policy/policy_oem_provisioning_test.json
#"$path_tools" -t psoc_c3 build-ramapp-package --app packets/apps/prov_oem/
#cyapp_prov_oem_signed_ifx0.bin --inparams packets/apps/prov_oem/in_params.bin --output packets/
#app/prov_oem/prov_oem_dlm.hex
load_image $image_file

# Notify ROM Boot that DLM package has been uploaded and ready for validation
mww $BOOT_DLM_CTL2 $IMAGE_SLOT_START_ADDR
mww $BOOT_DLM_CTL 0
puts "Waiting for application completion..."
sleep $wfa_img_verify_timeout ; # No debugger communication until DAP is re-enabled

# The BootROM disconnect debugger is after application was placed in RAM to
# update RAM protection settings. So, the BOOT_DLM_STATUS read operation fails.
# Wait until debugger is connected and BOOT_DLM_STATUS is changed.
set timeout 0
set dlm_status $CYBOOT_WFA_POLLING
while {$dlm_status == $CYBOOT_WFA_POLLING} {
    incr timeout 1
    if {$timeout > $wait_for_app_timeout} {
        if {$dlm_status == $CYBOOT_WFA_POLLING} {
            puts [format "FAIL: BootROM did not pass control to application - timeout, status =
0x%08X" $dlm_status]
        }

        exit $EXIT_DUE_ERROR
    }
    sleep $DEFAULT_TIME_OUT_MS

    # Set dlm_status previous value if read operation failed (debugger still disconnected)
    if { [catch { set dlm_status [mrw $BOOT_DLM_STATUS] }] } {
        set dlm_status $CYBOOT_WFA_POLLING
    }
}

# Wait for application completion if it was laucned by the BootROM
while {($dlm_status == $CYBOOT_SERVICE_APP_LAUNCHED)
    || ($dlm_status == $CYAPP_APP_RUNNING)} {
    incr timeout 1
    if {$timeout > $wait_for_app_timeout} {
        if {$dlm_status == $CYBOOT_WFA_POLLING} {
            puts [format "FAIL: BootROM did not pass control to application - timeout, status =
0x%08X" $dlm_status]
```

```
        } elseif {$dlm_status == $CYAPP_APP_RUNNING} {
            puts [format "FAIL: Application did not return - timeout, status = 0x%08X"
$dlm_status]
        } else {
            puts [format "FAIL: unexpected reason of faliure - timeout, status = 0x%08X"
$dlm_status]
        }
        exit $EXIT_DUE_ERROR
    }
    sleep $DEFAULT_TIME_OUT_MS

    set dlm_status [mrw $BOOT_DLM_STATUS]
}

# Application is either completed or was not launched
if {$dlm_status == $CYBOOT_SERVICE_APP_NOT_LAUNCHED} {
    puts [format "FAIL: BootROM did not launch application due verification failure, status =
0x%08X" $dlm_status]
} else {
    puts [format "Application execution completed status: 0x%08X" $dlm_status]
}

puts "\nSuccess"
local_echo off
catch { mww $RES_SOFT_CTL $RES_SOFT_CTL_RESET_RQST }
local_echo on
shutdown
```

***Note**: The default transport interface is set to KitProg3 in the TCL script. To change the interface to JLink, uncomment JLink.cfg and comment out `kitprog3.cfg` in the script*

4.4.2 Launch flow: OEM application

In this flow, the OEM application is combined with the DLM package to form a combined hex file. The DLM (Downloadable Module) is a hex package that includes an Infineon signed RAM_APP and the input parameters required for provisioning the device. The RAM_APP is responsible for provisioning these input parameters to the device's SFlash.

Flow details:

1. When the device powers on, the BootROM launches the OEM application
2. The OEM application does the following:
   a. Sets the boot DLM registers SRSS_BOOT_DLM_CTL to 0x6 and SRSS_BOOT_DLM_CTL2 to the SRAM address. This configuration tells the BootROM to validate the RAM_APP situated at the SRAM location from the SRSS_BOOT_DLM_CTL2 register
   b. Copies the DLM package from flash to the SRAM address
   c. Initializes the watchdog timer with a timeout interrupt (minimum timeout required is 32 ms)
   d. Puts the device into Deep Sleep with RAM retention enabled

3. Once the watchdog timer interrupt is triggered, the device exits deep sleep mode and follows the warm boot path. During warm boot, execution always starts from the BootROM
4. As part of warmboot sequence, the BootROM does the following:
   a. Checks for the SRSS_BOOT_DLM_CTL register to see if the RAM_APP is present and jumps to the address mentioned in SRSS_BOOT_DLM_CTL2. If any of the registers are not set, the BootROM continues with the normal boot to launch the OEM application
   b. Launches the RAM_APP to provision the input parameters
   c. Upon successful completion, the device sets the SRSS_BOOT_DLM_STATUS register to 0xF2A00001 indicating successful provisioning
   d. After successful provisioning, the BooTROM triggers a cold reset to reset the device and launch OEM application as part of normal boot process

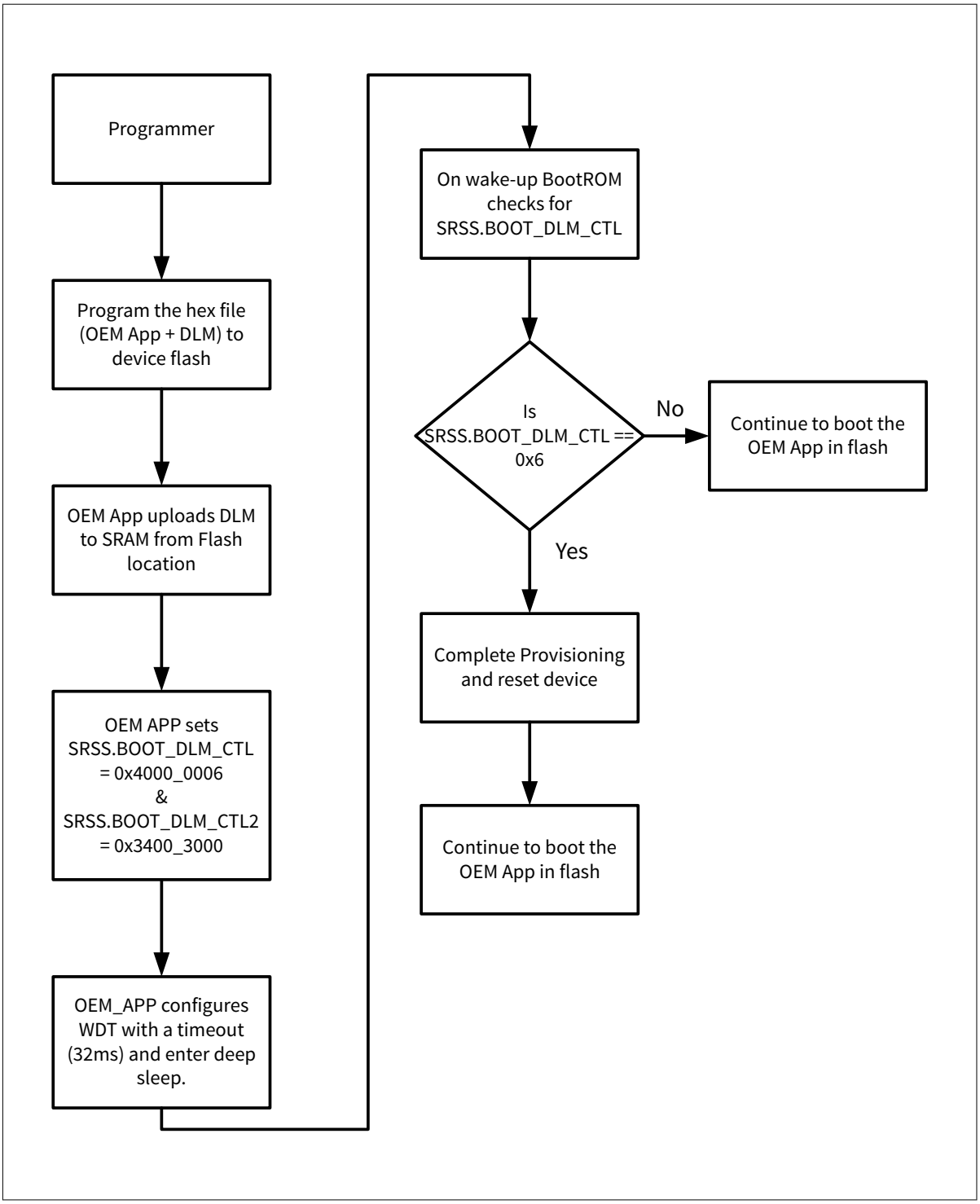


Figure 11 OEM application provisioning flow

Refer to PSOC™ Control C3 MCU: OEM Provisioning code example for the implementation.

4.4.3 Launch flow: BootROM DFU

Configuring the DFU Interface for Provisioning

The EdgeProtect Tools support two interfaces for device provisioning:

- OpenOCD (default)
- DFU

To use the DFU interface instead of OpenOCD:

- Configure the DFU Interface
- Execute provisioning

After setting DFU interface, use standard provisioning commands. BootROM will handle communication through DFU protocol, no additional configuration is needed.

5 Appendix

5.1 RAM application (DLM) status codes

The RAM application status code is written to the BOOT_DLM_STATUS register. This code is read by the Edge Protect Tools package to provide command execution logs.

Table 4 RAM application status codes

Code	Status name	Description
0x45000002UL	CYAPP_BAD_PARAM	One or more invalid parameters
0x45000003UL	CYAPP_LOCKED	Resource lock failure
0x45000004UL	CYAPP_STARTED	Operation started, but not necessarily completed yet
0x45000005UL	CYAPP_FINISHED	Operation finished
0x45000006UL	CYAPP_CANCELED	Operation canceled
0x45000007UL	CYAPP_TIMEOUT	Operation timed out
0x45000010UL	CYAPP_APP_RUNNING	The provisioning application is in progress
0x45000020UL	CYAPP_OTP_INIT_FAILED	Failed to initialize OTP
0x45000021UL	CYAPP_OTP_BOOTROW_WRITE_FAILED	Failed to update the LCS
0x45000022UL	CYAPP_OTP_BOOTROW_READ_FAILED	Failed to read the LCS
0x45000023UL	CYAPP_OTP_WRITE_FAILED	Failed to program object into the OTP
0x45000024UL	CYAPP_OTP_READ_FAILED	Failed to read object from the OTP
0x45000025UL	CYAPP_OTP_BOOTROW_CORRUPTED	BOOTROW is corrupted
0x45000026UL	CYAPP_NUM_ZEROS_INVALID	-
0x45000030UL	CYAPP_LCS_INVALID	The current device LCS is invalid for the execution of the RAM application
0x45000031UL	CYAPP_OEM_KEY_ALREADY_REVOKED	The OEM key 0 was revoked. This is operation can only be performed once
0x45000032UL	CYAPP_ICV_KEY_ALREADY_REVOKED	The ICV key 0 was revoked. This is operation can only be performed once
0x45000033UL	CYAPP_SIGNATURE_VERIF_FAILED	Failed to verify the input parameter's digital signature
0x45000034UL	CYAPP_KEY_0_ALREADY_PROGRAMMED	The OEM key 0 was programmed. This is operation can only be performed once
0x45000035UL	CYAPP_KEY_1_ALREADY_PROGRAMMED	The OEM key 1 was programmed. This operation can only be performed once
0x45000036UL	CYAPP_OEM_ASSETS_ALREADY_PROGRAMMED	The OEM assets were programmed. This operation can only be performed once
0x45000037UL	CYAPP_HASH_CALCULATION_FAILED	Hash calculation failed
0x45000038UL	CYAPP_INVALID_BOOT_CFG_ID	Invalid BOOT_CFG_ID configuration

(table continues...)

Table 4 (continued) RAM application status codes

Code	Status name	Description
0x45000039UL	CYAPP_HUK_GEN_FAILED	Failed to generate the Wrap Master Key
0X4500003AUL	CYAPP_DFU_FAILED	DFU flow failed
0X4500003BUL	CYAPP_PROT_FW_POLICY_DISABLED	Protected firmware policy is not configured
0X4500003CUL	CYAPP_MPC_POLICY_INVALID	MPC_POLICY is invalid
0X4500003DUL	CYAPP_OTP_NOT_DEFAULT	OTP assets are not zero
0X4500003EUL	CYAPP_BAD_APP_LAYOUT	Application start address and/or size is invalid
0X4500003FUL	CYAPP_PROT_FW_KEY_ALREADY_REVOKED	The PROT_FW key 0 was revoked. This is operation can only be performed once
0x4500003EUL	CYAPP_HASH_CALCULATION_FAILED	Failed to calculate hash
0x4500003FUL	CYAPP_PROVISIONING_LIMITS_FAILED	Provisioning limits exceeded
0x45000100UL	CYAPP_PARAM_NV_CNT_INVALID	The input parameter NV counter is not valid
0x450000101UL	CYAPP_PARAM_OEM_KEY_0_HASH_INVALID	The input parameter OEM key 0 hash is not valid
0x450000102UL	CYAPP_PARAM_OEM_KEY_1_HASH_INVALID	The input parameter OEM key 1 hash is not valid
0x450000103UL	CYAPP_PARAM_PUBKEY_INVALID	The OEM public key used for digital signature verification does not match the OEM public key hash provisioned in the OTP
0x450000104UL	CYAPP_PARAM_CONTROL_WORD_INVALID	The input parameter control word is not valid
0x450000105UL	CYAPP_PARAM_TARGET_LCS_INVALID	The input parameter target LCS is not valid. Valid values are NORMAL, NORMAL_NO_SECURE, and SECURE
0x450000106UL	CYAPP_RMA_CERT_VERIF_FAILED	Failed to verify the certificate for RMA LCS transition
0x450000107UL	CYAPP_PARAM_ACCESS_RESTRICT_INVALID	The input parameter ACCESS_RESTRICT has an invalid configuration
0x450000108UL	CYAPP_RMA_ATTEMPT_COUNTER_INVALID	RMA_ATTEMPT_COUNTER is equal to zero
0x450000109UL	CYAPP_IAK_ALREADY_PROGRAMMED	IAK already programmed
0x45000010AUL	CYAPP_BOOTROW_CHECK_FAILED	BootROW check failed
0x45000010BUL	CYAPP_EFUSE_CHECK_FAILED	eFuse data check failed
0x45000010CUL	CYAPP_CBOR_INVALID	CBOR packet is invalid
0X4500010DUL	CYAPP_PROT_FW_POLICY_INVALID	PROT_FW_POLICY is invalid after decryption

(table continues...)

Table 4 (continued) RAM application status codes

Code	Status name	Description
0xF2A00001UL	CYAPP_SUCCESS	The RAM application is completed successfully
0xF2A00010UL	CYAPP_APP_RUNNING	The RAM application is in progress

5.2 Edge Protect Tools supported commands

Table 5 List of supported commands for PSC3 target device

Command	Description
bin2hex	Converts a binary image to a hex image
build-ramapp-package	Builds the DLM package with the RAM application and input parameters
cbor2json	Converts CBOR to JSON format
convert-key	Converts key to other formats
cose-sign	Creates a COSE_Sign message
cose-sign1	Creates a COSE_Sign1 message
cose-verify	Verifies the COSE_Sign message
create-key	Creates private and public key pairs
create-provisioning-packet	Creates a binary packet for device provisioning
debug-token	Creates debug token binary-based on template
device-info	Reads device information - Silicon ID, Family ID, Silicon revision
device-list	Lists the supported devices
encrypt-aes	Encrypts binary files using AES encryption
hash	Calculates hash of a file
hex-segment	Extracts a segment from the hex. file
hex2bin	Converts hex image to binary image
init	Initializes a new project
integrity-exam	Executes the integrity exam app
merge-bin	Merges different binary files into one
merge-hex	Merges different hex files into one
provision-device	Executes device provisioning
read-die-id	Reads die ID from the device
reprovision-device	Executes device re-provisioning
rma-token	Creates 'transit to RMA' or 'open RMA' token
serial-config	Configures serial interfaces
serial-ports	Lists connected compatible hardware

(table continues...)

Table 5 (continued) List of supported commands for PSC3 target device

Command	Description
set-ocd	Sets on-chip debugger
transit-to-rma	Transitions device to RMA lifecycle stage
verify-image	Verifies image with a key
verify-packet	Verifies provisioning packet with a key
version	Shows firmware version

To view a list of options for a specific command, execute the following in the shell CLI:

```
edgeprotecttools -t psoc_c3 <COMMAND> --help
```

6 References

Contact Infineon Support to obtain these documents.

Application notes:

1. Infineon Technologies AG: AN238329: Getting started with PSOC™ Control C3 MCU on ModusToolbox™ software; Available online
2. Infineon Technologies AG: AN240106: Getting started with PSOC™ Control C3 security; Available online

Datasheets:

1. Infineon Technologies AG: 002-36422: PSOC™ Control C3 - PSC3P2xD, PSC3M3xD; Available online
2. Infineon Technologies AG: 002-36032: PSOC™ Control C3 - PSC3P5xD, PSC3M5xD; Available online

Reference manuals:

1. Infineon Technologies AG: PSOC™ Control C3 architecture reference manual; Available online
2. Infineon Technologies AG: PSOC™ Control C3 registers reference manual; Available online

For additional documents related to the PSOC™ Control C3 MCU, see the PSOC™ Control C3 MCU, PSOC™ Control C3P, and PSOC™ Control C3M webpages.

Glossary

ASF
Alternate serial interface

CA
Certificate authority

CBOR
Concise Binary Object Representation. See the CBOR webpage.

CLI
Command-line interface

COSE
CBOR Object Signing and Encryption

CoT
Chain of Trust

CSR
Certificate signing request

CWT
CBOR web token

DFU
Device Firmware Update

DFU HT
Device Firmware Update Host Tool

DLM
Downloadable module. Alternatively referred to in this document as RAM application

DLM package
Package consisting of the RAM application (DLM), input parameters, and the OEM signature

EAP
Early access pack

EPC 2
Edge Protect Category 2. See Infineon Edge Protect.

FW
Firmware

GPIO
General-purpose input/output

I2C
Inter-integrated circuit

Glossary

LCS
Lifecycle stage

MCU
Microcontroller unit

MFG
Manufacturing

MPC
Memory protection controller

NVM
Non-volatile memory

OEM
Original Equipment Manufacturer. Infineon's customers.

OEM_ROT_KEY
OEM Root of Trust key pair

OEM_ROT_PRIVATE_KEY
Private part of the OEM_ROT key pair.

OEM_ROT_PUBLIC_KEY
Public part of the OEM_ROT key pair.

OTA
Over-the-air

OTP
One-time programmable

PC
Personal computer

PEM
Privacy Enhanced Mail

PSA
Platform Security Architecture

RAM application
RAM applications are developed, signed by Infineon, and loaded into SRAM for the purpose of provisioning and LCS transition.

RMA
Return merchandise authorization

RoT
Root of Trust

Glossary

RRAM
Resistive RAM

RSA
Rivest–Shamir–Adleman

SFDP
Serial Flash Discoverable Parameter

SPE
Secure processing environment

SPI
Serial Peripheral Interface

SRSS
System resources subsystem

SWD
Serial Wire Debug

TLV
Tag-length-value

UART
Universal Asynchronous Receiver Transmitter

USB
Universal Serial Bus

Revision history

Document version	Date of release	Description of changes
**	2025-04-11	Initial release
*A	2025-10-17	Updated Debug policy Updated Edge Protect Tools Updated Launch flow: Debugger (SYS-AP)

Trademarks

PSOC™, formerly known as PSoC™, is a trademark of Infineon Technologies. Any references to PSoC™ in this document or others shall be deemed to refer to PSOC™.

Trademarks

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

Edition 2025-10-17

Published by

Infineon Technologies AG

81726 Munich, Germany

© 2025 Infineon Technologies AG

All Rights Reserved.

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

Email: erratum@infineon.com

Document reference
IFX-yuw1740386928967

Important notice

The information contained in this application note is given as a hint for the implementation of the product only and shall in no event be regarded as a description or warranty of a certain functionality, condition or quality of the product. Before implementation of the product, the recipient of this application note must verify any function and other technical information given herein in the real application. Infineon Technologies hereby disclaims any and all warranties and liabilities of any kind (including without limitation warranties of non-infringement of intellectual property rights of any third party) with respect to any and all information given in this application note.

The data contained in this document is exclusively intended for technically trained staff. It is the responsibility of customer's technical departments to evaluate the suitability of the product for the intended application and the completeness of the product information given in this document with respect to such application.

Warnings

Due to technical requirements products may contain dangerous substances. For information on the types in question please contact your nearest Infineon Technologies office.

Except as otherwise explicitly approved by Infineon Technologies in a written document signed by authorized representatives of Infineon Technologies, Infineon Technologies' products may not be used in any applications where a failure of the product or any consequences of the use thereof can reasonably be expected to result in personal injury.