Please note that Cypress is an Infineon Technologies Company.

The document following this cover page is marked as “Cypress” document as this is the company that originally developed the product. Please note that Infineon will continue to offer the product to new and existing customers as part of the Infineon product portfolio.

Continuity of document content

The fact that Infineon offers the following product as part of the Infineon product portfolio does not lead to any changes to this document. Future revisions will occur when appropriate, and any changes will be set out on the document history page.

Continuity of ordering part numbers

Infineon continues to support existing part numbers. Please continue to use the ordering part numbers listed in the datasheet for ordering.

CE222802 – PSoC 6 MCU Encrypted Bootloader

Objective

This example demonstrates PSoC® 6 MCU bootloading with applications that are encrypted or have encrypted digital signatures.

Requirements

Tool: PSoC Creator 4.2; Peripheral Driver Library (PDL) 3.0.3 with Bootloader SDK 2.20; OpenSSL 1.0.2 or later; Python 2.7 or later

Programming Language: C (Arm® GCC 5.4.1 and Arm MDK 5.22)

Associated Parts: All PSoC 6 MCU parts

Related Hardware: CY8CKIT-062-BLE PSoC 6 BLE Pioneer Kit

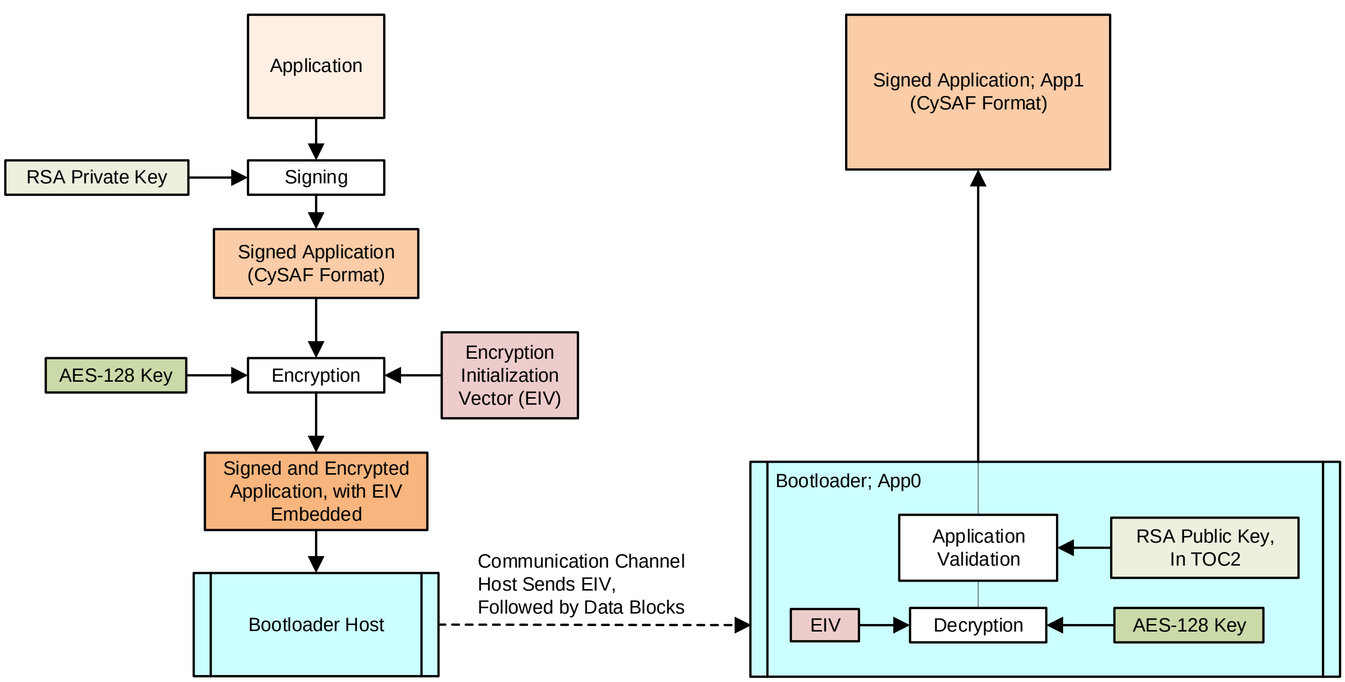
Overview

This example demonstrates a simple UART-based bootloading system that is similar to that in CE213903, *PSoC 6 MCU Basic Bootloaders*; with additional features – signing and encryption – needed for secure bootloading, as Figure 1 shows.

Note: To facilitate testing, the secure bootloading features are demonstrated with the device in the Normal life cycle stage. This means that the overall system is not secure. To create a secure system, the device must be placed in the Secure life cycle stage. For more information, see AN221111, *PSoC 6 MCU Creating a Secure System*.

This example has two applications in PSoC 6 MCU flash: "Bootloader_Encrypted_App0" and "Bootloader_Encrypted_App1", or just "App0" and "App1" for short. Each application is a separate PSoC Creator project. App0 is the bootloader application; it downloads, installs, and transfers control to the bootloadable application (App1). Memory Layout shows where App0 and App1 are located in flash.

Figure 1. Signed and Encrypted Bootloader Process Flow Diagram



Signing: This feature prevents bootloading unauthorized applications. The bootloadable application is built in the Cypress Secure Application Format (CySAF), which includes a signature that is encrypted using the private key in an RSA[1] key pair. Note that the signature is the only portion of the application that is encrypted. The corresponding public key is placed in the Table of Contents 2 (TOC2) in PSoC 6 MCU supervisory flash (SFlash). Using the public key, the bootloader decrypts the signature and validates the bootloadable application in flash before transferring control to it, as Figure 1 shows. For more information, see AN221111.

Note: The bootloader project is also built in the CySAF format. Using the same public key in TOC2, the bootloader is validated by a Flash Boot module in SFlash, as part of the boot sequence that takes place at device initialization. If validation fails, the bootloader is not executed. This example includes a method to disable bootloader validation for development purposes.

Encryption: This feature prevents IP theft by "sniffing" the communication channel during application download. The application is encrypted before downloading, and decrypted by the bootloader before installing it in device flash. The application is encrypted using the AES-128-CBC [2] algorithm, which uses symmetric keys for encryption and decryption. That is, the same key exists in both the development environment and the bootloader. This method works if the communication channel is the only part of the system that can be accessed, for example by monitoring BLE over-the-air (OTA) traffic. If access to the key is gained, the transmissions can be decrypted and the application IP is vulnerable to theft.

It is possible to implement the above features separately; they each address a different security issue. This code example demonstrates how the features are combined to address both issues at the same time.

Hardware Setup

No special hardware setup needs to be done for the CY8CKIT-062-BLE PSoC 6 BLE Pioneer Kit. Connect the kit's USB port to your computer's USB port. The KitProg2 system on the kit acts as both a programmer for direct programming, and as a USB-UART bridge for UART bootloading. For more information, see the KitProg2 User Guide.

Software Setup

To customize the bootload operation and enable Bootloader SDK features, update the `#define` statements as needed in the *bootload_user.h* file. This design uses a specific feature set to enable validation and to enhance security. For more information, see File Modifications.

Operation

OpenSSL and Python are required to generate key files, and to create and sign downloadable application files. Install OpenSSL 1.0.2 or later and Python 2.7 or later in your system.

OpenSSL offers source files for download, which need to be compiled. A list of sites offering OpenSSL binaries can be found here. Make sure that the system Path variable in your computer includes the OpenSSL binary folder. Changes to the system Path variable may require a computer restart to take effect.

Generate New Keys

This example includes default key files. These files are used to sign and validate (RSA key pair), and encrypt and decrypt (AES key), the bootloadable application. You can build and test the code example projects without changing the default keys.

To generate new keys, do the following steps; note that all of the default files are overwritten.

Note: For security reasons, the AES encryption initialization vector (EIV) for the CBC[3] operation mode should be a random number and never used more than once. You should regenerate the EIV for each revision of your bootloadable application.

1. Run the *keygen.bat* batch file located in the project folder *CE222802_Bootloader_Encrypted_App0.cydsn*. Confirm that a new folder *keys_generated* is created, and contains multiple key text files.

[1] RSA: RSA Laboratories has defined public key-based encryption/decryption and digital signature schemes called RSAES PKCS and RSASSA PKCS, respectively.
[2] AES: Advanced Encryption Standard. AES-xxx refers to the number of bits used, for example AES-128, AES-192 or AES-256. Increasing the number of bits increases bootloading time; see Number of Bits in AES Encryption.
[3] CBC: Cipher block chaining, an AES encryption/decryption algorithm.

2. Run the *key_copy.bat* batch file located in the project folder *CE222802_Bootloader_Encrypted_App0.cydsn*. Press the 'Y' key to continue. The batch file renames and copies the generated key files to the workspace *CE222802_Keys.cylib* folder, overwriting the existing files.

3. Open the *rsa_to_c_generated.txt* file in the *keys_generated* folder. Copy its contents into the *cy_si_keystorage.c* file in the project folder *CE222802_Bootloader_Encrypted_App0.cydsn*. A comment in the *cy_si_keystorage.c* file indicates where to paste the data.

4. Open the *aes_private_array_generated.txt* file in the *keys_generated* folder. Copy its contents into the *bootload_user.c* file in the project folder *CE222802_Bootloader_Encrypted_App0.cydsn*, overwriting the existing array `AES128_Key[]`.

Build the Projects

Note: The *CE222802_Keys.cylib* folder and the *.txt* files therein, as mentioned in Generate New Keys, are included in the workspace for bundling and distribution purposes only. They are not directly used in either project.

1. Using PSoC Creator, build the App0 project. For more information on device programming, see PSoC Creator Help. The *post_build_core1.bat* file in this project generates a signed bootloader *.hex* file.

   Build the App1 project. The *post_build_core1.bat* file in this project generates a signed and encrypted application *.cyacd2* file.

   Note: During the build process, you may be prompted to replace files in the project with files from the PDL. The PDL files are templates. Do not replace the customized files in the project. Click **Cancel**.

2. Connect CY8CKIT-062-BLE PSoC 6 BLE Pioneer Kit to your computer using the USB cable.

3. Program the App0 project into the kit. For more information on device programming, see PSoC Creator Help.

   Note: Programming App0 into the PSoC 6 MCU includes adding data to TOC2. This limits the device to execute only the signed App0 code from this example. To enable unsigned code, see Disable Code Validation.

4. Confirm that the red LED on the kit blinks once every two seconds. This indicates that App0 has been validated and is running.

5. Press and hold the kit button for at least half a second, and then release it. Confirm that nothing happens because App0 is the only application installed.

Download the Encrypted Application

1. Select **Tools** > **Bootloader Host...**.to run PSoC Creator Bootloader Host Program (BHP).

   Establish a connection with your computer's COM port corresponding with the KitProg2 USB-UART bridge, using the UART configuration from App0 (Baud: 115200; data bits: 8; stop bits: 1; no parity).

   For more information on using BHP, see BHP Help or the Bootloader SDK User Guide.

2. Browse and select the *CE222802_Bootloader_Encrypted_App1_encrypted.cyacd2* file, located in the project folder *CE222802_Bootloader_Encrypted _App1 \ CortexM4 \ [compiler name] \ Debug*. Click **Open**.

3. Select **Actions** > **Program** to download App1. After the download is complete, confirm that the kit red LED blinks twice per second, indicating that App1 has been validated and is running.

4. Press and hold the kit button for at least half a second, and then release it. Confirm that the kit red LED blinks once per two seconds, indicating that control has been transferred back to App0.

5. Repeat steps 1 – 4 to test the process of reinstalling App1.

6. While in App0 and not updating App1, press and hold the kit button for at least half a second, and then release it. Confirm that the kit red LED blinks twice per second, indicating that control has been transferred to App1.

Disable Code Validation

As part of the PSoC 6 MCU boot process, the Flash Boot module validates App0, using the public key in TOC2 and the encrypted signature in App0. If you reprogram the device to replace App0 with another project, the device will not boot. If you want to reprogram the device with other projects, you must first disable the validation of App0 – do the following steps. Note that validation of App1 is also disabled.

1. In the *main_cm0p.c* file of App0, set the `#define UNLOCK_SYSTEM` statement to a nonzero value.

2. Build App0 and program it into the PSoC 6 MCU.
3. Confirm that the red LED on the kit remains ON, indicating that the device can now run unsigned code.

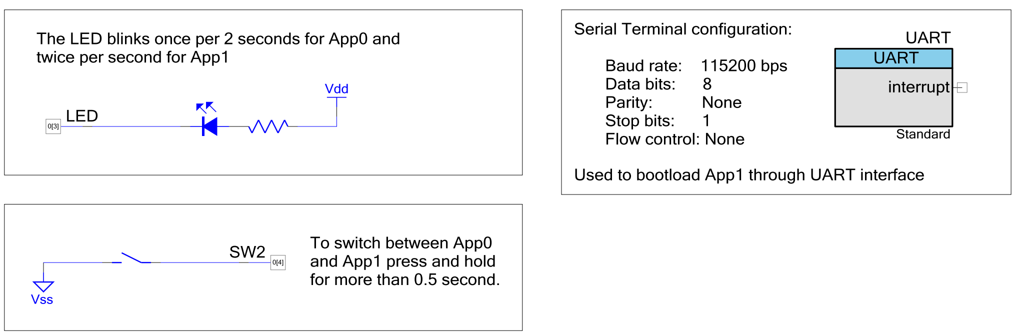
Design and Implementation

This example has two applications, called “App0” and “App1”. Each application is a separate PSoC Creator project with the following features:

- App0 is the bootloader application; it downloads, installs, and transfers control to the bootloadable application (App1).
- Both applications are signed and encrypted.
- App1 is decrypted by App0 and stored in a temporary location in user flash for validation. App0 then validates App1 and, after validation is successfully completed, moves App1 to its proper location and transfers control to App1.
- Each application blinks the kit red LED at a different rate, making it easy to see which application is currently running.
- Holding a kit button down for more than half a second, then releasing it, causes the application that is currently running to transfer control to the other application.

Figure 2 shows the PSoC Creator project schematic for both App0 and App1. App0 has the host communication Component; App1 does not.

Figure 2. PSoC Creator Schematic for App0 and App1



Design Firmware

Application Validation

This code example shows how to configure a PSoC Creator project to create an application that can be validated.

In order to be validated, an application must be signed. Both applications in this example are in the CySAF format, which includes a signature field. The signature in that field is a hash of the application; the hash is encrypted using an RSA private key.

The corresponding public key is saved in TOC2. At device startup, the Flash Boot module uses the public key to validate App0 before transferring control to it. App0 uses the same public key to validate App1 after downloading it and before transferring control to it.

In this example, the TOC2 contents, including the public key, are defined in the *main_cm0p.c* file in the app0 project. When the app0 project is programmed into the PSoC 6 MCU device, TOC2 is initialized with the public key and the address of app0.

This example includes batch files and Python programs to generate the public and private keys for application validation, as well as a key for encryption of App1. See Generate New Keys.

Firmware Files

The firmware portion of the design is implemented in the files listed in Table 1 (for App0) and Table 2 (for App1).

Table 1. Design Firmware Files for Bootloader App0

File	Description
main_cm4.c / main_cm0p.c	Contains the `main()` function for each CPU. PSoC 6 MCU has two CPUs: An Arm® Cortex®-M4 (CM4) and a Cortex-M0+ (CM0+). See Table 4 for specific tasks for each core. The *main_cm0p.c* file contains the CySAF header for this application.
cy_bootload.h /.c	The bootloader software development kit (SDK) files
bootload_user.h	Contains user-editable `#define` statements that control the operation and enabled features in the SDK
bootload_user.c	Contains user functions required by the SDK: • Five functions that control communications with the bootloader host. These are also called “transport functions”. • Two functions – `ReadData()` and `WriteData()` – that control access to internal or external memory In this example, this file also has a `Cy_Bootload_DecryptData()` function to decrypt the incoming application.
transport_uart.h /.c	Contains bootloader transport functions for the host communications Component being used. These functions are typically called by the transport functions in *bootload_user.c*.
bootload_common.ld	GCC linker script. It is user-editable – it controls the memory layout and the locations in memory for each application, and the code and data for each CPU core in each application. This file is included in the custom GCC linker scripts described next. This file is common to all applications.
bootload_cm4.ld, *bootload_cm0p.ld*	Custom GCC linker scripts. In each application, these files replace the auto-generated linker script files. These files locate the code and data sections for each of the CPU cores as well as the bootloader and other regions. These files include the memory layout described in *bootload_common.ld*.
bootload_mdk_common.h *bootload_mdk_symbols.c*	Similar in function to the GCC and IAR common linker scripts, for MDK. The MDK linker does not support includes in *.scat* files, so these files exist to create the necessary defines. These files are user-editable – they control the memory layout and the locations in memory for each application, and the code and data for each CPU core in each application. These files are common to all applications.
bootload_cm4.scat, *bootload_cm0p.scat*	Custom MDK linker scripts. In each application, these files replace the auto-generated linker script files. These files locate the code and data sections for each of the CPU cores as well as the bootloader and other regions.
bootload_common.icf	IAR linker script. It is user-editable – it controls the memory layout and the locations in memory for each application, and the code and data for each CPU core in each application. This file is included in the custom IAR linker scripts described below. This file is common to all applications.
bootload_cm4.icf, *bootload_cm0p.icf*	Custom IAR linker scripts. In each application, these files replace the auto-generated linker script files. These files locate the code and data sections for each of the CPU cores as well as the bootloader and other regions. These files include the memory layout described in *bootload_common.icf*.
cy_si_config.h	Contains definitions required for setting up the TOC2 and the application headers
cy_si_keystorage.c /.h	Contains definitions for key storage and the arrays that hold the keys
post_build_core1.bat	Batch file to sign App0

Table 2. Design Firmware Files for Bootloadable App1

File	Description
main_cm4.c / main_cm0p.c	Contains the `main()` function for each CPU. PSoC 6 MCU has two CPUs: An Arm Cortex-M4 (CM4) and a Cortex-M0+ (CM0+). See Table 4 for specific tasks for each core. The *main_cm0p.c* file contains the CySAF header for this application.
cy_bootload.h /.c	The bootloader software development kit (SDK) files. These are included in this application solely for enabling transfer of control to another application.
post_build_core1.bat	Batch file to sign and encrypt App1.

File Modifications

Note: The buffer to store bootloader host commands, in `main()` in the *main_cm4.c* file in App0, was greatly increased in size as a temporary fix for a possible attack vector using the set app metadata command with Bootloader SDK 2.10, which could result in a buffer overflow and execution of malicious code. This attack vector can also be fixed by disabling the set app metadata command – set the `CY_BOOTLOAD_METADATA_WRITABLE` macro in *bootload_user.h* to zero. A proper fix will be included in the next release of the Bootloader SDK.

To enable validation and encryption, the following macros in the *bootload_user.h* file in App0 are set to a nonzero value: `CY_BOOTLOAD_OPT_SET_EIVECTOR` and `CY_BOOTLOAD_OPT_CRYPTO_HW`. Also:

- `CY_BOOTLOAD_APP_FORMAT` is set to `CY_BOOTLOAD_CYPRESS_APP`. This enables validation of App1 when it is in CySAF format.
- `CY_BOOTLOAD_SEC_APP_VERIFY_TYPE` is set to `CY_BOOTLOAD_VERIFY_FULL`. This enables application, key, and TOC checks.

To enhance security and privacy, the following macros in the *bootload_user.h* file in App0 are set to zero: `CY_BOOTLOAD_OPT_VERIFY_DATA`, `CY_BOOTLOAD_OPT_ERASE_DATA`, `CY_BOOTLOAD_OPT_GET_METADATA`. This disables support of the corresponding host commands.

The `CY_BOOTLOAD_MAX_APPS` macro is set to '3' to enable storing downloaded applications in a temporary location for validation. The `AppID` of App1 is set to '2'. To change the temporary location, modify the `temporaryLocation` variable in the *bootload_user.c* file in App0.

The signature size defined in the common linker script is changed from 4 to 256 bytes to accommodate the RSA signature.

To decrypt the incoming data, the Crypto engine library is included in the project's build settings (see **Peripheral Driver Library** in the Build Settings dialog. The `Cy_Bootload_DecryptData()` function in *bootload_user.c* in App0 decrypts incoming data before writing it into the flash memory.

Post-build batch files are added to both applications to populate the signature with a unique hash using the private RSA key. Additionally, the post-build batch file in App1 generates an encrypted downloadable application file using the AES-128-CBC algorithm with the AES key and EIV. See Figure 1 on page 1.

Memory Layout

Figure 3 shows the typical memory usage for each CPU in each application. This layout is for the signed bootloader using UART as the communication channel in PSoC 6 MCU devices with 1 MB flash and 288 KB SRAM.

To change the memory layout or usage, update the linker script files shown in Table 1.

Figure 3. Memory Usage of App0 and App1

	Address	Description	Size
USER FLASH	0x100F FC00	Metadata copy row	512 B
	0x100F FA00	Metadata row	512 B
	0x1006 0000	Unused	767 KB
	0x1005 0000	App1 Signature App1, CM4	64 KB
	0x1004 0000	App1, CM0+ App1 CySAF Header	64 KB
	0x1002 0000	Unused	128 KB
	0x1001 0000	App0 Signature App0, CM4	64 KB
	0x1000 0000	App0, CM0+ App0 CySAF Header	64 KB

	Address	App0 and App1	Size
RAM	0x0804 7FFF 0x0800 A000	Unused	248 KB
	0x0800 2000	CM4 Application Data	32 KB
	0x0800 0100	CM0+ Application Data	7.75 KB
	0x0800 0000	Common RAM	256 B

Components and Settings

Table 3 lists the PSoC Creator Components used in this example, how they are used in the design, and the non-default settings required so they function as intended.

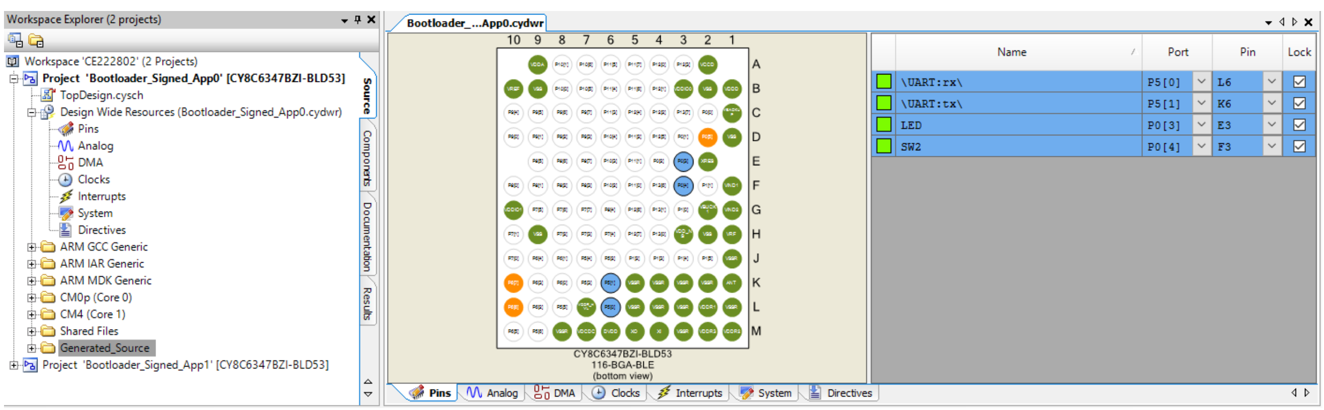
Table 3: PSoC Creator Components

Component	Instance Name	Purpose	Non-default Settings
UART (SCB)	UART	Host communication	(none)
Digital Output Pin	LED	Drive an LED	No HW connection; External terminal; Initial drive state High (1); Max frequency 1 MHz
Digital Input Pin	SW2	Read button state	No HW connection; External terminal; Initial drive state High (1); Max frequency 1 MHz

Design-Wide Resources

Figure 4 shows the pin assignments for the CY8CKIT-062-BLE PSoC 6 BLE Pioneer Kit, for the UART, LED, and user button.

Figure 4. Pin Assignments for CY8CKIT-062-BLE PSoC 6 BLE Pioneer Kit for the Signed Bootloader



Reusing This Example

Note: The App0 and App1 projects must be built with the same toolchain (GCC or MDK), or application transfer may fail. Check the **Build Settings** for each project.

This code example is easily portable to the CY8CKIT-062 PSoC 6 Pioneer Kit. This kit has the same pin assignments for the LEDs, button, and communication channels as CY8CKIT-062-BLE. Change the device to CY8C6247BZI-D54.

Dual Core

PSoC 6 MCU has two CPU cores: Cortex-M4 and a Cortex-M0+. An application can include code for one or both cores. For more information, see AN215656, *PSoC 6 MCU Dual-CPU System Design*.

In these examples, CPUs in each application do as Table 4 shows. This can easily be changed so that either CPU can run any of the tasks, including bootloading.

Table 4. CPU Tasks in Each Application

Application	Cortex-M0+	Cortex-M4
Bootloader (App0)	• Executes first at device reset. Reset handler controls application transfer. • Turns ON CM4 • Processes Crypto server requests	• Blinks an LED once per two seconds • Bootloads App1 • Monitors the button • After bootload or when button pressed, initiates transfer of control to App1, with software reset
User Application (App1)	• Executes first, then turns ON CM4 • Does nothing else	• Blinks an LED twice per second • Monitors the button • When button is pressed, initiates transfer of control to App0, with software reset

Software Reset

When transferring control from one application to another, the recommended method is through a device software reset. This enables each application to initialize device hardware blocks and signal routing from a known state.

It is possible to freeze the state of I/O pins so that they are maintained through a software reset. Defined portions of SRAM are also maintained through a software reset. For more information, see the *PSoC 6 MCU: PSoC 63 with BLE Architecture Technical Reference Manual*.

Number of Bits in AES Encryption

In this example, App1 is encrypted using AES-128, that is, 128-bit encryption. It is possible to change the number of bits to 192 or 256. This increases encryption robustness at the cost of increased bootloading time. The following instructions show how to change to 256-bit; the instructions for 192-bit are similar:

1. Change the *post_build_core1.bat* file in App1: `--encrypt AES-128-CBC` to `–encrypt AES-256-CBC`.
2. Change the *keygen.bat* file in App0: in the line `openssl rand -hex -out %OUT_DIR%\%AES_TEMP% 16`, change the ‘`16`’ to ‘`32`’.
3. Follow the instructions in Generate New Keys, with the following changes.
   a. When you copy the *aes_private_array_generated.txt* file to the *bootload_user.c* file in App0, change the array `AES128_Key[16]` to `AES128_Key[32]`. Changing the array name is recommended but optional.
   b. Also in *bootload_user.c*, in the line:
   ```
   cryptoStatus = Cy_Crypto_Aes_Init((uint32_t *)AES128_Key, CY_CRYPTO_KEY_AES_128, &AES_context);
   ```
   change the parameter ‘`CY_CRYPTO_KEY_AES_128`’ to ‘`CY_CRYPTO_KEY_AES_256`’.

Related Documents

PSoC 6 MCU Bootloader-Related Application Notes	
AN213924 – PSoC 6 MCU Bootloader Software Development Kit (SDK) Guide	Provides information on how to use the Bootloader SDK, as well as information on bootloading in general.
Other Application Notes	
AN210781 – Getting Started with PSoC 6 MCU with Bluetooth Low Energy (BLE) Connectivity	Describes PSoC 6 MCU with BLE Connectivity devices and how to build your first PSoC Creator project
AN215656 – PSoC 6 MCU: Dual-CPU System Design	Describes the dual-CPU architecture in PSoC 6 MCU, and shows how to build a simple dual-CPU design
AN221111 – PSoC 6 MCU: Creating a Secure System	Describes how to build a secure embedded system with a PSoC 6 MCU
PSoC 6 MCU Bootloader-Related Code Examples	
CE213903 – PSoC 6 MCU Basic Bootloaders	Describes a basic bootloader using UART, I^2C, or SPI.
CE221984 – PSoC 6 MCU Dual-Application Bootloader	Describes a basic bootloader with two applications
CE216767 – PSoC 6 MCU with Bluetooth Low Energy (BLE) Connectivity Bootloader	Describes a BLE bootloader
CE220959 – PSoC 6 MCU with BLE Bootloader Using External Memory	Describes a BLE bootloader that uses SMIF external memory
CE220960 – PSoC 6 MCU BLE Upgradeable Stack Bootloader	Describes a BLE bootloader with an upgradeable BLE stack

PSoC Creator Component Datasheets	
UART	Supports the serial communication block in UART mode
Pins	Supports connection of hardware resources to physical pins

Device Documentation	
PSoC 6 MCU: PSoC 63 with BLE Datasheet	PSoC 6 MCU: PSoC 63 with BLE Architecture Technical Reference Manual

Development Kit Documentation
CY8CKIT-062-BLE PSoC 6 BLE Pioneer Kit

Document History

Document Title: CE222802 – PSoC 6 MCU Encrypted Bootloader

Document Number: 002-22802

Revision	ECN	Orig. of Change	Submission Date	Description of Change
**	6135484	MKEA	04/12/18	New code example

Worldwide Sales and Design Support

Cypress maintains a worldwide network of offices, solution centers, manufacturer's representatives, and distributors. To find the office closest to you, visit us at Cypress Locations.

Products

Arm® Cortex® Microcontrollers	cypress.com/arm
Automotive	cypress.com/automotive
Clocks & Buffers	cypress.com/clocks
Interface	cypress.com/interface
Internet of Things	cypress.com/iot
Memory	cypress.com/memory
Microcontrollers	cypress.com/mcu
PSoC	cypress.com/psoc
Power Management ICs	cypress.com/pmic
Touch Sensing	cypress.com/touch
USB Controllers	cypress.com/usb
Wireless Connectivity	cypress.com/wireless

PSoC® Solutions

PSoC 1 | PSoC 3 | PSoC 4 | PSoC 5LP | PSoC 6 MCU

Cypress Developer Community

Community | Projects | Videos | Blogs | Training | Components

Technical Support

cypress.com/support

All other trademarks or registered trademarks referenced herein are the property of their respective owners.

Cypress Semiconductor
198 Champion Court
San Jose, CA 95134-1709