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.

AN228680

Secure System Configuration in Traveo™ II Family

Author: Koji Mizumoto

Associated Part Family: Traveo™ II Family

Related Application Notes: see Related Documents

This document explains what is required to configure a secure system from the boot process to application software execution in Traveo II family.

Contents

1 Introduction

This application note discusses how to make sure that the system executes code only from a trusted source and how to configure a secure embedded system using Traveo II family MCU. You will learn about the boot process and how it pertains to a secure system.

This is an advanced application note and assumes that you are familiar with the basic Traveo II architecture (see the Device Datasheet or the Architecture Technical Reference Manual (TRM)).

There are many reasons why a system must be secure: manufacturers want to protect their IP to maintain their market, or to protect the end user from dangerous operations caused by a malicious attack from a third party.

There are three main ways products can be hacked:

- **Direct access to the debug port:** Traveo II is based on Arm® architecture; accessing or reprogramming firmware or examining internal data is easy with the use of a common debugger. Hacking or reverse engineering a product is easy if the device is left unsecured.
- **Direct connection to a communication port such as SPI, I²C, CAN, LIN, or a UART:** These ports are used for communication between ECUs, or ECU and other components in the car. It is also used for ECU software update or information acquisition. This connection may allow firmware to be read or updated with non-sanctioned access if the device is left unsecured.
- **Wireless connection such as OTA:** This has become the standard method of hacking because it doesn't require physical contact. Perpetrators can access the car to perform unauthorized firmware update and control from anywhere if the device is left unsecured.

2 What is a Secure System?

The definition of a secure system can be different depending on the application. Some systems require that all access to the device is blocked, but others just need to verify that the firmware has not been corrupted. Traveo II MCU allows you to define the security level required for the project. There is no one perfect method because every project has different requirements. The following is a list of projects with different security goals:

- **Trusted firmware updates only with a hardware debugger:** This is usually not thought of as a secure system, but if the hardware is installed such that third parties cannot get direct access, then it may be secure. Flash write commands from firmware can be disabled so that any internal hack could not change or replace the application. The device can be put in a secure mode with the debug port open, which will force the firmware to be authenticated with a public key each time the device comes out of reset.
- **No access to debug port, support firmware updates:** The debug port provides access to all memory and a method for the device to be reprogrammed. In most cases, a real secure system requires that the debug port is disabled. Then, the only way to update the firmware is for the application to provide a way to download new program data with some type of communication port such as UART, I²C, CAN, LIN, or SPI. How secure this communication port must be is up to the designer.
- **Lock down firmware, no updates:** This means that the debug port is disabled and there is no provision for bootloading. This may be the most secure, but there is no way to perform bug fixes or add future enhancements.
- **Trusted firmware updates, protect IP:** To fully protect the IP, the debug port must be disabled. Because the debug port is disabled, the user must provide a method to load new firmware with a bootloader. This is typically implemented with a serial port such as UART, SPI, or I²C. Because the IP must be protected, the bootloader must encrypt the data transferred. Traveo II MCU includes a crypto block to help accelerate encryption and decryption. Security keys installed in the device at the factory can be used to authenticate the code and decrypt the data transferred during the bootload process.

3 Basic Definitions

This section describes some terms that will be used throughout this document.

- **Application Flash (User)**: This is the flash memory that is used to store your application code.

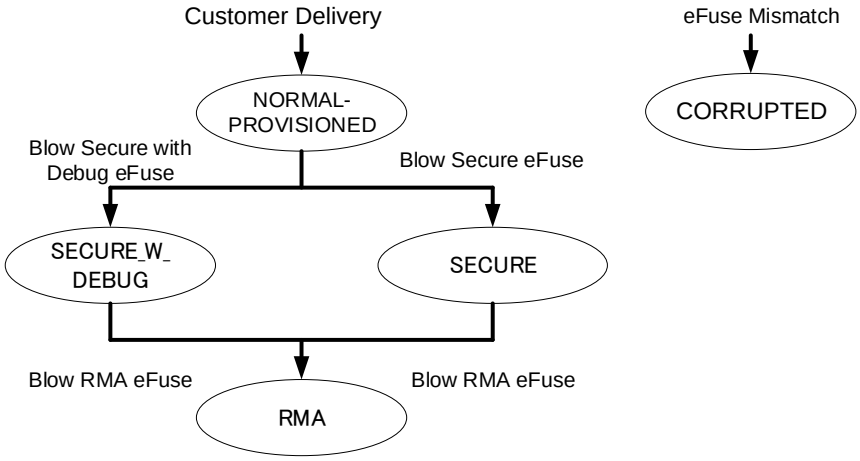
- **Chain of Trust (CoT)**: The root of trust begins with the Cypress code residing in the ROM, which cannot be altered. Chain of Trust is established by validating the blocks of software before the execution starting from the root of trust located in the ROM.
- **Cipher-based Message Authentication Code (CMAC)**: Message authentication code algorithm based on block cipher. (e.g., AES)
- **Debug Access Port (DAP**): Interface between an external debugger/programmer and Traveo II MCU for programming and debugging. This allows the connection to one of the three access ports (AP), CM0_AP, CM4_AP or CM7_AP, and System_AP. The System_AP can only access SRAM, flash, and MMIOs, not the CPUs.
- **DAP Access Restriction**: This determines the debug port access restrictions, and has three states corresponding to the protection state: Normal, Secure, and Dead. Each of these states may be configured by the user. The Normal protection state is stored in SFlash, but Secure and Dead state configurations are stored in the one-time programmable eFuse.
- **Digital Digest/Signature**: The signature generated by the SHA-256 function that operates on a block of data.
- **Electronic Control Unit (ECU)**: Unit for controlling the system using electronic circuits; mainly mounted on automobiles. There are various ECUs depending on the application, such as body control, engine control, and brake control.
- **eFuse**: One Time Programmable (OTP) memory, that by default is '0' and can be changed only from '0' to '1'. The eFuse bits can be programmed individually, but cannot be erased.
- **eFuse Read Protection Unit (ERPU)**: This is part of SWPU (Software Protection Unit, see definition below). ERPUs are used to implement read access restrictions to eFuse.
- **eFuse Write Protection Unit (EWPU)**: This is part of SWPU. EWPUs are used to implement write access restrictions to eFuse.
- **Flash Boot**: This is part of the boot system that performs two basic tasks:
  - Sets up the debug port based on the lifecycle stage.
  - Validates the user application before executing it.
- **Flash Write Protection Unit (FWPU)**: This is part of SWPU. FWPUs are used to implement write access restrictions to flash.
- **Hash**: A crypto algorithm that generates a repeatable but unique digest for a given block of data. This function is non-reversible.
- **IP**: Intellectual Property. This can be both code and data stored in a device.
- **IPC**: Inter-Processor Communication hardware used to facilitate communication between the two CPU cores.
- **Lifecycle**: This is the security mode in which the device is operating. Traveo II has five stages NORMAL_PROVISIONED, SECURE, SECURE_W_DEBUG, RMA, and CORRUPTED. To the user, it has only three states of interest: NORMAL_PROVISIONED, SECURE, and SECURE_W_DEBUG.
- **Main User Application**: This is part of the user application that is not authenticated by flash boot. It is mainly executed by the CM4 or CM7 CPU. For CoT, it needs to be authenticated with the Secure Image.
- **Memory Protection Unit (MPU)**: MPU is used to isolate memory sections from different software components executed on the same CPU. MPU is bus-master-specific.
- **MMIO:** Memory-Mapped Input/Output, usually refers to registers that control the hardware I/O.
- **Non-Secure (NS):** NS is a protection attribute used to distinguish between secure and non-secure accesses. In non-secure setting, secure access is also allowed. The NS attribute is allowed and restricted by SMPU, PPU, and SWPU.
- **Over the Air (OTA):** OTA refers to transmission and reception of data via wireless communication.
- **Protection Context (PC):** PC can apply different protection attributes to bus master access without changing the setting of protection units. The PC attribute is allowed and restricted by SMPU, PPU, and SWPU. Although PC most often refers to a Program Counter, in this document, it refers to the Protection Context.
- **Peripheral Protection Unit (PPU)**: PPUs are used to restrict access to a peripheral or set of peripherals to only one or a specific set of bus masters.
- **Protection State**: There are Normal, Secure, Dead, and Virgin states depending on the lifecycle stage. ROM boot deploys access restrictions according to the protection status.

- **Protection Units**: These are four types of protection units: Memory Protection Unit (MPU), Shared Memory Protection Unit (SMPU), Peripheral Protection Unit (PPU), and Software Protection Units (SWPU). MPU, SMPU, and PPU are implemented by hardware; SWPU is implemented by software.
- **Public-Key Cryptography (PKC)**: Otherwise known as asymmetrical cryptography. Public-key cryptography is an encryption technique that uses a paired public and private key (or asymmetric key) algorithm for secure data communication. It is used to decode a message or block of data. The private key is used to decrypt the data and must be kept secured, while the public key is used to encrypt the data but can be disseminated widely.
  - **Public Key**: The public key can be shared, but it should be authenticated or secured so it cannot be modified.
  - **Private Key**: The private key must be kept in a secure location so it cannot be viewed or stolen. It is used to decrypt a block of data that has been encrypted using an associated public key.
- **RMA:** Returned Material Authorization
- **ROM**: Read Only Memory that is programmed as part of the fabrication process and cannot be reprogrammed.
- **RSA-nnnn**: An asymmetric encryption system that uses two keys. One key is private and should not be shared and the other is public and can be read without loss of security. The encryption/decryption is controlled by a key that is commonly 2048, 3072, or 4096 bits in length (RSA-2048, RSA-3072, or RSA-4096).
- **Secure Image**: A software that is used to set up the security features such as HSM firmware of Traveo II MCU; it is mainly executed by CM0+. It can be modified by the programmer to implement a specific security policy.
- **Security Policy**: This is the set of rules that the designer imposes to determine what resources are protected from outside tampering or between the internal CPUs.
- **Serial Memory Interface (SMIF)**: An SPI (Serial Peripheral Interface) communication interface to serial memory devices, including NOR flash, SRAM, and non-volatile SRAM.
- **SFlash**: Supervisor flash memory. This memory partition in flash contains several areas that include system trim values, flash boot executable code, public key storage, etc. After the device transitions into a SECURE lifecycle stage, it can no longer be changed.
- **SHA-256**: A cryptographic hash algorithm used to create a digest for a block of data or code. This hash algorithm produces a 256-bit unique digest of the data no matter the size of the data block.
- **Shared Memory Protection Unit (SMPU)**: SMPUs are used to allow access to a specific memory space (flash, SRAM, or registers) to only one or a specific set of bus masters.
- **Software Protection Units (SWPU)**: SWPUs are used to implement access restrictions to flash write, and eFuse read and write.
- **System Calls**: Functions such as flash write functions that are executed by the Arm® Cortex® M0+ CPU (CM0+) from ROM.
- **TOC1**: This is an area in SFlash that is used to store pointers to the trim values, flash boot entry points, etc. It is used only by boot code in ROM and is not editable by the designer.
- **TOC2**: This is an area in SFlash that is used to store pointers to two applications blocks: Secure Image and Main User Application. It also contains some boot parameters that are settable by the system designer.

4 Lifecycle Stage

The device lifecycle is a key aspect of the Traveo II MCU security. Lifecycle stages follow a strict irreversible progression dictated by blowing eFuses (changing a fuse's value from '0' to '1'). This system is used to protect the internal device data and code at the level required by the customer. Figure 1 shows Traveo II MCU-supported lifecycle stages.

Figure 1. Device Lifecycle Stages



4.1 NORMAL_PROVISIONED

This is the lifecycle stage of a device after trimming and testing is complete in the factory. All configuration and trimming information is complete. Valid flash boot code has been programmed in the SFlash. To allow the OEM to check the data integrity of trims, flash boot, and other objects from the factory, a hash (SHA-256 truncated to 128 bits) of these objects is stored in eFuse. This hash is referred to as FACTORY_HASH. Customers receive parts in this stage.

4.2 SECURE

This stage is the life cycle stage of a secure device. Prior to transitioning to this stage, the SECURE_HASH must have been blown in eFuse and valid application code must have been programmed in the code flash. In this stage, the protection state is set to SECURE and SECURE access restrictions are deployed. A SECURE device will boot only when authentication of its flash boot and application code succeeds. The SECURE_HASH is calculated and written to the eFuse by SROM firmware, when transitioning to the SECURE or SECURE_W_DEBUG lifecycle stage from the NORMAL_PROVISIONED lifecycle stage.

4.3 SECURE_W_DEBUG

This stage is same as SECURE lifecycle stage, except the device allows for debugging. Prior to transitioning to this stage, the SECURE_HASH must have been blown in eFuse and valid application code must have been programmed in the code flash. In this stage, the protection state is set to SECURE, but NORMAL access restrictions are deployed to enable debugging. When there is an authentication failure during ROM boot or flash boot, the protection state is set to “SECURE” but NORMAL access restrictions are deployed, SWD/JTAG pins are enabled, and is not transitioned to “DEAD” state. Transition from SECURE_W_DEBUG to SECURE is not allowed. SECURE_W_DEBUG parts are used only by the developers and SW testers. Devices should not be shipped in this lifecycle stage.

4.4 RMA

This stage allows one to perform Failure Analysis (FA). The customer transitions the part to RMA lifecycle stage when the customer wants Cypress to perform failure analysis on the part. The customer erases all the sensitive data prior to invoking the system call that transitions the part to RMA.

When invoking the system call to transition to RMA, the customer must create a certificate that authorizes to transition the part with a specific Unique ID to RMA lifecycle stage. The certificate will be signed by the customer using the same private key that is used in signing the user application image. The verification of the signature uses exactly the same algorithm used by flash boot in authenticating the user application. The same public key (injected by the OEM) stored in SFlash is used for the verification.

When a part is reset in the RMA lifecycle stage, the boot will set the access restrictions such that the DAP has access only to System AP, IPC MMIO registers for making system calls, and adequate RAM for communication. It will then wait for the system call “Open RMA” from the DAP along with the certificate of authorization. The boot process will not initiate any firmware until it successfully executes the “Open RMA” command. After the command is successful, the lifecycle stage stored in eFuse cannot be changed from RMA. Every time the part is reset, it must execute the “Open RMA” command successfully before the part can be used. To execute "Open RMA", the customer must provide the certificate signed by the customer using their private key to Cypress.

Note that the "Open RMA" system call is executed by Cypress using the provided certificate.

4.5 CORRUPTED

The device is in this lifecycle stage if read error is detected when reading the eFuse bits that determine the lifecycle stage. The device will enter DEAD protection state and only IPC MMIOs can be read via SYS-AP. No other accesses are allowed.

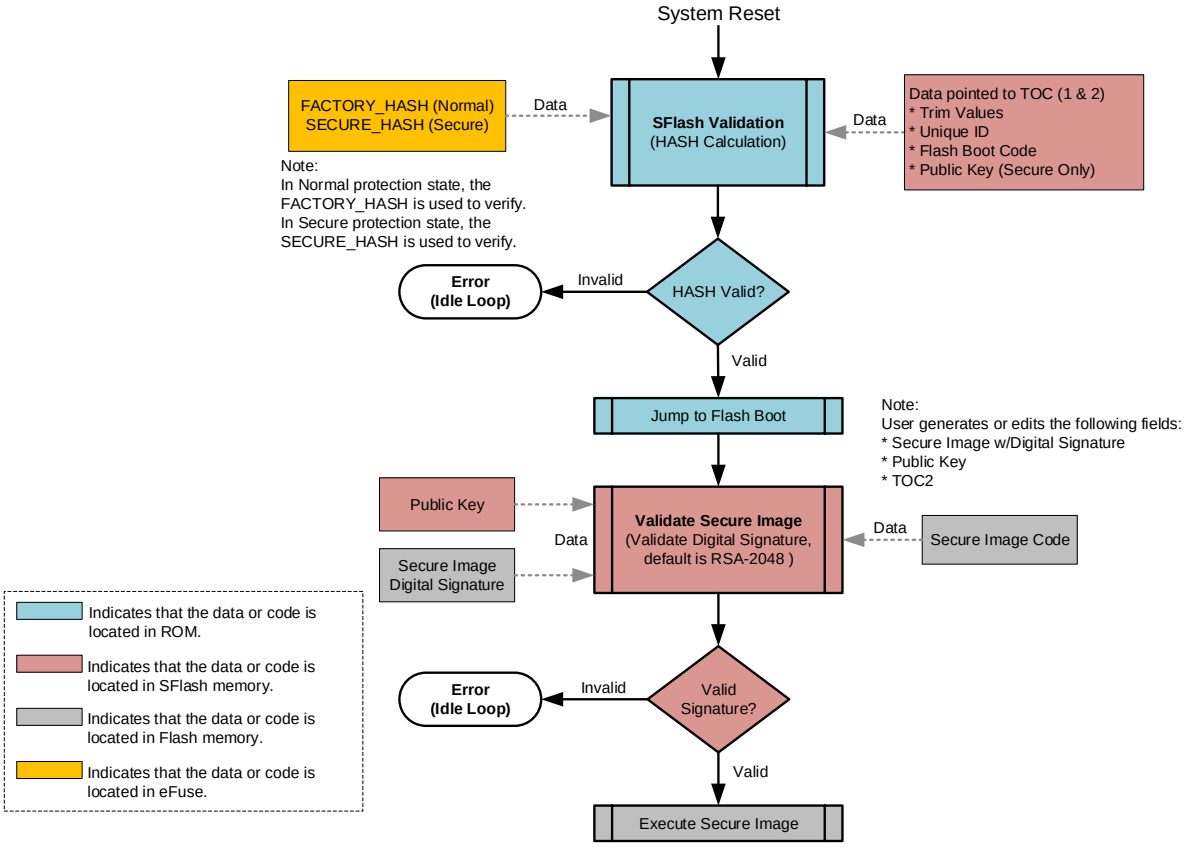
5 Boot Sequence and Chain of Trust

The CoT is inherently part of the boot sequence. It begins at the root of trust, which is the initial boot code stored in ROM, and which cannot be changed by users or Cypress.

5.1 Boot Sequence

The boot sequence and the validation sequence are, for the most part, one and the same. Figure 2 shows how the CM0+ operation starts from reset. After reset, the CM0+ starts executing from ROM boot. ROM boot validates SFlash. After validation of SFlash is complete, execution jumps to flash boot and configures the DAP as needed by the protection state. Notice the color coding that depicts the memory type where the data and code resides.

Figure 2. Traveo II MCU Boot Sequence with CoT



The flash boot then validates the first application listed in the TOC2 and jumps to its entry point if validated. In the secure system defined in this application note, the first user application is the Secure Image. After the Secure Image configures the hardware to secure the system, it will validate the Main User Application, if needed.

If the secure image is found to be invalid or corrupted, the CPU will jump to an idle loop and stay in the idle loop until the device is reset.

5.2 Chain of Trust (CoT)

The basis of the chain of trust relies on the memory that cannot be changed, such as ROM. The rest of the chain is dependent on this fact.The ROM code cannot be changed and is used to validate the next block of execution; in this case it is the flash boot.

Flash boot code, trim constants, and the TOC1 are located in SFlash that cannot be reprogrammed. Most of it is preprogrammed at the factory and the calculated hash value for this section is stored in eFuse and referred to as the FACTORY_HASH value. This ensures that the flash boot code, trim values, and the TOC1 has not been tampered with after the MCU has left Cypress.

After the lifecycle stage transition from Normal to Secure, the SFlash blocks are validated with another hash value referred to as the “SECURE_HASH”. This value is also stored in eFuse and cannot be changed after it is programmed. These items in SFlash include the TOC2 and the public key.

The FACTORY_HASH value that is used to validate the SFlash in the Normal lifecycle stage is stored in eFuse and cannot be changed. A different location is used to validate the SFlash with extra items that were programmed, and SECURE_HASH. This location is written into a separate section of eFuse.

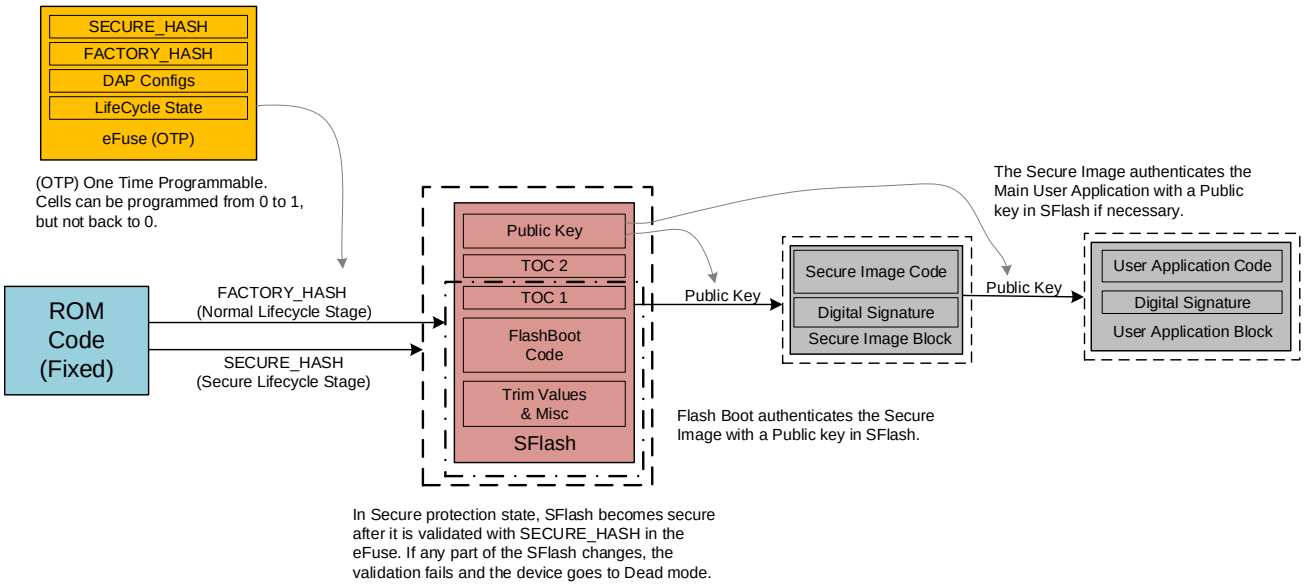
The entire SFlash block is validated with the SECURE_HASH each time the device wakes from reset in the Secure lifecycle stage. If an error is found while validating the SFlash, the device will no longer complete the boot sequence, and enter a Dead state.

When verification is successful, the entire SFlash is now trusted because its validation is based on memory (eFuse) that cannot be modified without detection during SFlash validation in ROM.

The public key, which is locked into the SFlash, is secure and cannot be changed without being detected as well. It is used by flash boot to validate the next step in the boot process. The flash boot validates the code in the Secure Image block, which includes a digital signature at the end of the code block. The flash boot uses the SHA-256 hash function to calculate the digital signature of the Secure Image Block. The digital signature attached to the Secure Image block is encrypted using a private key that is associated with the Public Key in SFlash using RSA 2048-bit encryption. The calculated and the stored encrypted digital signatures are then checked to see whether they match. If they match, the Secure Image Block has been validated. The same process can be used by the Secure Image to validate the User Application Block. See Figure 4 and Figure 5 for the signing and validation flow of Secure Image code.

Figure 3 shows the CoT from the perspective of data and code validation.

Figure 3. Basic CoT



6 Code Signing and Validation

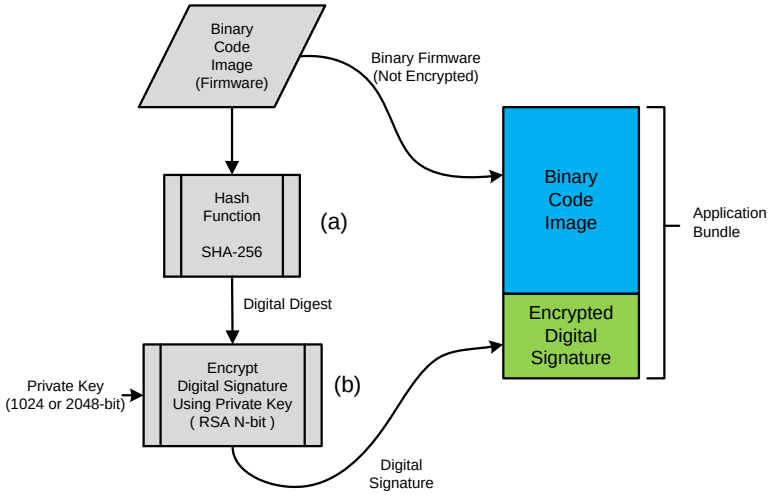
This section describes the process of signing a block of code so that it can be validated during boot time.

The encryption method used is Public Key Cryptography (PKC) that uses a private and public key. Care must be taken to keep the private key at a secure location, so that it never gets into the public domain. If the private key is exposed, it will endanger your system's security. Companies must create a method in which very limited access to the private key is allowed. The public key, on the other hand, can be viewed by anyone. The only requirement is that the public key must be validated or locked in such a way that it can't be changed, or so that any modification to the public key can be detected. In Traveo II MCU, the public key is stored in SFlash and validated with the SECURE_HASH as defined in the CoT section above. More details of generating and using the private and public keys are discussed in Appendix A.

6.1 Code Signing

To validate code, such as the user applications during boot time, a digital signature must be created and bundled with the code during build time. The code itself is not stored in flash in an encrypted format but the digital signature is encrypted. The digital signature is generated with the SHA-256 hash function (a), then encrypted using a private key with RSA-2048-bit encryption (b). The reason the digital signature is encrypted is so that a third party, without access to the private key, cannot create a valid code/signature bundle. See Figure 4.

Figure 4. Generation of Encrypted Digital Signature



6.2 Code Validation

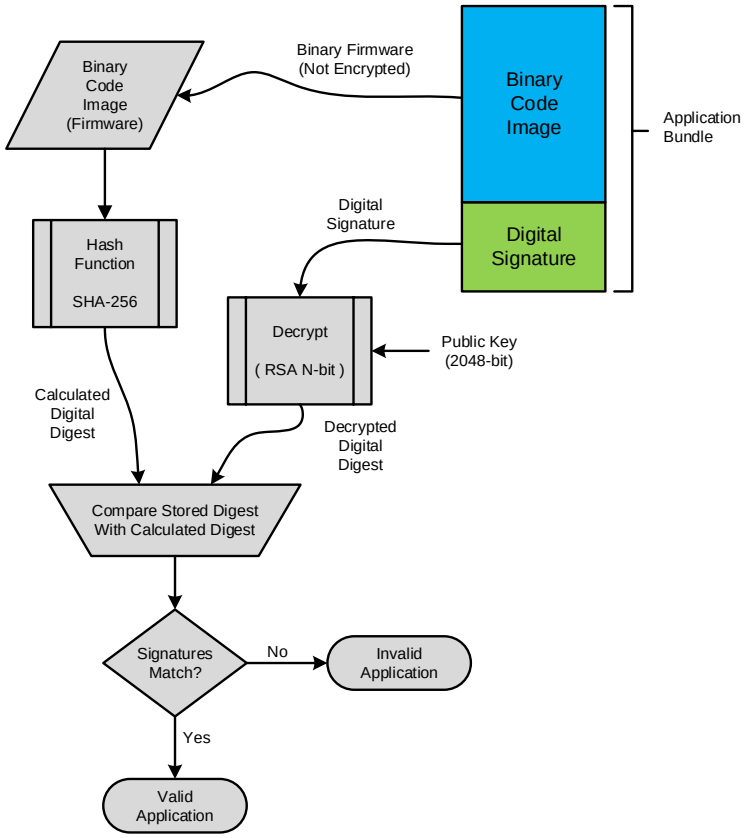
A secure system must be able to detect code that was not created by the original manufacturer or trusted source. If non-trusted code is detected, execution must take a known path to a safe state. This also validates that the firmware was not corrupted intentionally or accidently.

Validation requires three pieces: data, signature, and crypto key. The data and the signature come as a pair; the key is stored in a location that cannot be changed.

- **Data:** This includes both executable code and constants that make up the firmware in an embedded system. This data usually resides in flash memory that usually can be modified at one time or another. Depending on the system, you may modify or update it. Therefore, you must be able to determine whether this data is from a known source and has not been corrupted either by accident or by a malicious event.
- **Digital Signature:** A digital signature is the hash of a block of data. The hash algorithm used in this case is SHA-256. The digital signature alone can be used to validate that the data is intact. By encrypting the digital signature with a crypto key, you can determine whether the data is from a trusted source, as well as intact.
- **Crypto Key:** This can be either a public or private key. In the system described in this application note, the public key is stored on the device and the private key is secured by the developer. The public key must be secured in one of two ways:
    1. *A method to validate the source of the key.* This can be accomplished with some type of communication with a known source or server. This is not practical for devices that cannot easily communicate with a known server when required.
    2. *Have the key data itself validated by using a key that is stored in memory that cannot be modified*. This is the most likely option for an embedded system. In Traveo II MCU, a hash is calculated from the areas containing the public key, flash boot code, and trim values. This hash is then stored in one-time programmable eFuse and referred to as “SECURE_HASH”.

A user application binary code block (Application Bundle) includes an encrypted digital signature that was created during build time. The Secure Image application also uses this format. To validate the block of code, a hash function (SHA-256) is applied to the binary code image, which creates a calculated digital signature, or digital digest. Next, the Encrypted Digital Signature is decrypted using the stored public key, to reveal the decrypted digital signature. The calculated digital digest and the decrypted digital digest (signature) are then checked for an exact match. If they are an exact match, the code is validated. See Figure 5.

Figure 5. Code Validation



7 Resource Protection

Resource protection means that during runtime, only the bus master or task that should have access to a memory space or register space can access it. This can be a combination of read, write, or execute. Traveo II MCU have blocks called “protection units” to add this functionality. These protection units can be configured to create multiple protection zones that include flash, SRAM, peripherals, and I/Os. These zones can then be restricted by CPU, tasks, or both. There are four main types of protection units: MPU, SMPU, PPU and SWPU.

- Each CPU has its own MPU. The MPU is different from the protection structure of the other protection units. MPU protection structures cater only to protection attributes pertaining to a single master. MPU protection structures do not have a Protection Context parameter associated with them. Protection attributes for an MPU are user/privilege, read/write/execute. MPUs are specific for each bus master and provide resource protection from its various threads or tasks.
- SMPUs protect the memory regions that are used by multiple masters. These SMPUs have all the attributes of the MPU, the protection context, and non-secure attribute. The Secure Image uses SMPUs to restrict access to secure sections of the memory from the non-secure application. Registers used for flash write operations are restricted so that only the SROM code may access those operations. This eliminates any accidental writing or erasure of the flash memory. Also, SRAM and registers used for Crypto operations are protected to keep operations secure.
- PPUs are designed specifically to protect peripheral registers. A PPU is similar to the SMPU. The PPUs are fixed-function because they are hardwired to protect a specific peripheral region. Therefore, this type of PPU cannot configure address and size parameters in the protection structure, but you can set protection attributes.
- SWPUs are used to implement access restrictions to flash (program/erase) and eFuse (Read/Write). The SWPU is broken into two parts and stored in SFlash. The first part is configured by boot process, and cannot change. The second part can be used by the application for additional access restrictions specific to the application. It can be

updated in the NORMAL_PROVISIONED lifecycle stage by writing to specific row in SFlash. Also, it can also be updated using the SROM API. ROM/flash boot reads and configures the two parts of SWPU from SFlash. SWPU consists of FWPU, ERPU, and EWPU.

7.1 Boot Protection

Some protection units are configured during the boot process and cannot be reconfigured. These protection units are vital to providing a secure system and providing a reliable access to system call functions. See the BootROM chapter in Architecture TRM for details of Boot Protection.

7.2 Application Protection

You can configure access protection to flash and eFuse during the boot process using SWPU. This is called application protection units. Application protection units can be configured in SFlash with the chip in the NORMAL_PROVISIONED lifecycle state. The Application protection SWPU has up to 16 entries of FWPU and up to 4 entries of ERPU and EWPU. See the “Protection Unit” chapter in the Architecture TRM for details of SWPU. Table 1 shows the default value of Application Protection.

Table 1. SWPU Default Value of Application Protection

SWPU Layout	Description	Default Value
PU_OBJECT_SIZE (4 bytes)	SWPU Object Size	0x00000030
N_FWPU (4bytes) Max 16 entries	Number of FWPUs	0x00000000
N_ERPU	Number of ERPUs	0x00000001
ERPU0_SL_OFFSET (4 bytes)	Protection Offset Address Setting	0x00000068
ERPU0_SL_SIZE (4 bytes)	Region Size and ERPU0 Enable	0x80000018
ERPU0_SL_ATT (4 bytes)	Slave Attribute	0x00FF0007
ERPU0_SL_ATT (4 bytes)	Master Attribute	0x00FF0007
N_EWPU	Number of ERPUs	0x00000001
EWPU0_SL_OFFSET (4 bytes)	Protection Offset Address Setting	0x00000068
EWPU0_SL_SIZE (4 bytes)	Region Size and ERPU0 Enable	0x80000018
EWPU0_SL_ATT (4 bytes)	Slave Attribute	0x00FF0007
EWPU0_SL_ATT (4 bytes)	Master Attribute	0x00FF0007

See Appendix F to learn how to configure Application Protection.

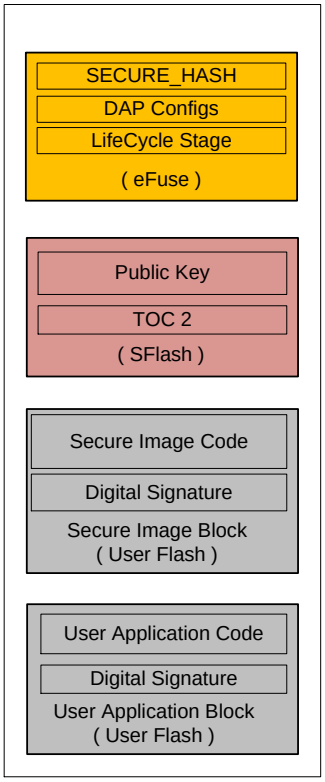
8 Configure a Secure System

This section will take you through each of the pieces of a secure system and describe how to generate them.

Configuring a fully secure system with a CoT is more complicated than generating a simple application. Instead of just the user code, it needs to contain several other pieces normally not required in a simple (non-secure) system. The following are the memory sections that will need to be programmed when creating a secure system:

- SECURE_HASH (eFuse)
- DAP Configs (eFuse)
- Lifecycle Stage (eFuse)
- Public Key (SFlash)
- TOC2 (SFlash)
- Secure Image Block (User flash)
- User Application Block (User flash)

Figure 6. Secure System Configuration



Building one block may be dependent on building another one. For example, TOC2 contains the start address of both the secure image block and the user application block. The SECURE_HASH stored in eFuse is dependent on everything in SFlash.

Note that some of these items are programmed implicitly through the appropriate SROM firmware.

8.1 TOC2

There are two table sections in SFlash: TOC1 and TOC2. TOC1 is used for internal purposes and is not user-editable. TOC2 is used to point to the location of each application. In TOC2, the Secure Image is indicated as the "User application" and the Main User Application is indicated as "CM4 or CM7 core x User application". (x is shows the CPU number.)

There are two parameters that you may want to change for a faster boot sequence, at offset 0x1F8 (See Table 2): the boot clock frequency parameter “IMO/FLL clock frequency” and the debug parameter “Wait Window Time”. You may change these in TOC2 to the values defined in the table. Table 2 lists the elements of TOC2.

Table 2. Elements of TOC2

Offset	Purpose	Default Value
0x00	Object size in bytes for CRC calculation starting from offset 0x00	0x00001FC
0x04	Magic number (Fixed: 0x01211220)	0x01211220
0x08	Null-terminated table of pointers representing the SMIF configuration structure	0x00000000
0x0C	Address of First User Application Object	0x10000000
0x10	Format of First User Application Object. 0: Basic, 1: Cypress standard, 2: Simplified	0x00000000 Set to 1, when Secure Boot
0x14	Address of Second User Application Object Second user application is validated if the first application validation failed	0x00000000

<table>
<tr><th>Offset</th><th>Purpose</th><th>Default Value</th></tr>
<tr><td>0x18</td><td>Format of Second User Application Object.
0: Basic, 1: Cypress standard, 2: Simplified</td><td>0x00000000
Set to 1, when Secure Boot</td></tr>
<tr><td>0x1C</td><td>Address of First CM4 or CM7 core1 User Application Object</td><td>0x00000000</td></tr>
<tr><td>0x20</td><td>Address of Second CM4 or CM7 core1 User Application Object</td><td>0x00000000</td></tr>
<tr><td>0x24</td><td>Address of First CM4 or CM7 core2 User Application Object</td><td>0x00000000</td></tr>
<tr><td>0x28</td><td>Address of Second CM4 or CM7 core2 User Application Object</td><td>0x00000000</td></tr>
<tr><td>0x100</td><td>Number of additional objects to be verified for SECURE_HASH</td><td>0x00000003</td></tr>
<tr><td>0x104</td><td>Address of signature verification key (0 if none).
The object is signature specific key. It is the public key in case of RSA.</td><td>0x00000000
Case of location in SFlash: 0x17006400</td></tr>
<tr><td>0x108</td><td>Address of Application Protection</td><td>0x17007600</td></tr>
<tr><td>0x10C-
0x1F0</td><td>Additional objects if needed, or 0s if none</td><td>0x00000000</td></tr>
<tr><td>0x1F4</td><td>TOC2 REVISION</td><td>-</td></tr>
<tr><td>0x1F8</td><td>Controls default configuration
Bits [1:0]: CLOCK_CONFIG
Flag to indicate clock frequency configuration. The clock should stay the same after flash boot execution.
<table>
<tr><th>Value [1:0]</th><th>Description</th></tr>
<tr><td>0x0</td><td>8 MHz, IMO, no FLL</td></tr>
<tr><td>0x1</td><td>25 MHz, IMO + FLL</td></tr>
<tr><td>0x2</td><td>50MHz, IMO + FLL (Default)</td></tr>
<tr><td>0x3</td><td>Use ROM boot clocks configuration (100 MHz)</td></tr>
</table>
Bits [4:2]: LISTEN_WINDOW
Flag to determine the Listen window to allow sufficient time to acquire debug port.
<table>
<tr><th>Value [4:2]</th><th>Description</th></tr>
<tr><td>0x0</td><td>20 ms (Default)</td></tr>
<tr><td>0x1</td><td>10 ms</td></tr>
<tr><td>0x2</td><td>1 ms</td></tr>
<tr><td>0x3</td><td>0 ms (No Listen window)</td></tr>
<tr><td>0x4</td><td>100 ms</td></tr>
<tr><td>Others</td><td>Reserved</td></tr>
</table>
Bits [6:5]: SWJ_PINS_CTL
Flag to determine if SWJ pins are configured in SWJ mode by flash boot.
Note: SWJ pins may be enabled later in the user code.
<table>
<tr><th>Value [6:5]</th><th>Description</th></tr>
<tr><td>0x0</td><td>Do not enable SWJ pins in Flash boot. Listen window is skipped.</td></tr>
<tr><td>0x1</td><td>Do not enable SWJ pins in Flash boot. Listen window is skipped.</td></tr>
<tr><td>0x2</td><td>Enable SWJ pins in Flash boot (default)</td></tr>
<tr><td>0x3</td><td>Do not enable SWJ pins in Flash boot. Listen window is skipped.</td></tr>
</table>
Bits [8:7]: APP_AUTH_CTL
Flag to determine if the application image digital signature verification (authentication) is performed:
<table>
<tr><th>Value [7:6]</th><th>Description</th></tr>
<tr><td>0x0</td><td>Authentication is enabled (default).</td></tr>
</table></td><td>0x00000242</td></tr>
</table>

<table>
<tr><th>Offset</th><th>Purpose</th><th>Default Value</th></tr>
<tr><td></td><td>
<table>
<tr><td>0x1</td><td>Authentication is disabled.</td></tr>
<tr><td>0x2</td><td>Authentication is enabled (recommended).</td></tr>
<tr><td>0x3</td><td>Authentication is enabled.</td></tr>
</table>

Bits [10:9]: FB_BOOTLOADER_CTL
Flag to determine if the internal bootloader in flash boot is disabled:
<table>
<tr><th>Value [9]</th><th>Description</th></tr>
<tr><td>0x0</td><td>Internal bootloader is disabled.</td></tr>
<tr><td>0x1</td><td>Internal bootloader is launched if other bootloader conditions are met (default).</td></tr>
<tr><td>0x2</td><td>Internal bootloader is disabled.</td></tr>
<tr><td>0x3</td><td>Internal bootloader is disabled.</td></tr>
</table>
</td><td></td></tr>
</table>

To generate proper values for the TOC2, an instance of the following code can be included in the project.

```
/** Flashboot parameters */
#define CY_SI_FLASHBOOT_FLAGS ((CY_SI_FLASHBOOT_CLK_50MHZ << CY_SI_TOC_FLAGS_CLOCKS_POS) \
                             | (CY_SI_FLASHBOOT_WAIT_20MS << CY_SI_TOC_FLAGS_DELAY_POS) \
                             | (CY_SI_FLASHBOOT_SWJ_ENABLE << CY_SI_TOC_FLAGS_SWJEN_POS) \
                             | (CY_SI_FLASHBOOT_VALIDATE_NO << CY_SI_TOC_FLAGS_APP_VERIFY_POS) \
                             | (CY_SI_FLASHBOOT_FBLOADER_DISABLE << CY_SI_TOC_FLAGS_FBLOADER_ENABLE_POS))
/** TOC2 in SFlash */
CY_SECTION(".cy_toc_part2") __USED static const cy_stc_si_toc_t cy_toc2 =
{
 .objSize        = CY_SI_TOC2_OBJECTSIZE,          /* Offset+0x00: Object Size (Bytes) excluding CRC */
 .magicNum       = CY_SI_TOC2_MAGICNUMBER,         /* Offset+0x04: TOC2 ID (magic number) */
 .smifCfgAddr    = 0UL,                            /* Offset+0x08: SMIF config list pointer */
 .cm0pappAddr1   = CY_SI_SECURE_FLASH_BEGIN,       /* Offset+0x0C: App1 (CM0+ First User App Object) addr */
 .cm0pappFormat1 = CY_SI_APP_FORMAT_CYPRESS,       /* Offset+0x10: App1 Format */
 .cm0pappAddr2   = CY_SI_USERAPP_FLASH_BEGIN,      /* Offset+0x14: App2 (CM0+ Second User App Object) addr */
 .cm0pappFormat2 = CY_SI_APP_FORMAT_CYPRESS,       /* Offset+0x18: App2 Format */
 .cm4_71appAddr1 = CY_SI_CM471_1stAPP_FLASH_BEGIN, /* Offset+0x1C: App3 (CM4/CM7_1 1st User App Object) addr */
 .cm4_71appAddr2 = CY_SI_CM471_2ndAPP_FLASH_BEGIN, /* Offset+0x20: App4 (CM4/CM7_1 2nd User App Object) addr */
 .cm72appAddr1   = CY_SI_CM72_1stAPP_FLASH_BEGIN,  /* Offset+0x24: App5 (CM7_2 1st User App Object) addr */
 .cm72appAddr2   = CY_SI_CM72_2ndAPP_FLASH_BEGIN,  /* Offset+0x28: App6 (CM7_2 1st User App Object) addr */
 .reserved1      = 0UL,                            /* Offset+0x2C-0xFF: Reserved area 212Bytes */
 .shashObj       = 0UL,                            /* Offset+0x100: Number of verified additional objects (S-HASH)*/
 .sigKeyAddr     = CY_SI_PUBLIC_KEY,               /* Offset+0x104: Addr of signature verification key */
 .swpuAddr       = CY_SI_SWPU_BEGIN,               /* Offset+0x108: Addr of SWPU Objects */
 .toc2Addr       = CY_SI_TOC2_BIGIN,               /* Offset+0x10C: Addr of TOC2 */
 .addObj         = 0UL,                            /* Offset+0x110-0x1F4: Reserved area 232Bytes */
 .tocFlags       = CY_SI_FLASHBOOT_FLAGS,          /* Flashboot flags stored in TOC2 */
 .reserved3      = 0UL,                            /* Offset+0x1FC: Reserved area 1Byte */
};
```

8.2 User Application Block

Secure applications must be validated with a public Crypto key. To do this, the application must use the Cypress secure application format that includes a digital signature. This allows flash boot to perform the validation during the boot process, before the application is executed. The application format encapsulates the application binary, application metadata, and an encrypted digital signature. Although there is a place for both CM0+ and CM4 images in this format, the Secure Image requires only the CM0+ image. The user application includes both images, see Figure 7.

Figure 7. Cypress Secure Application Format

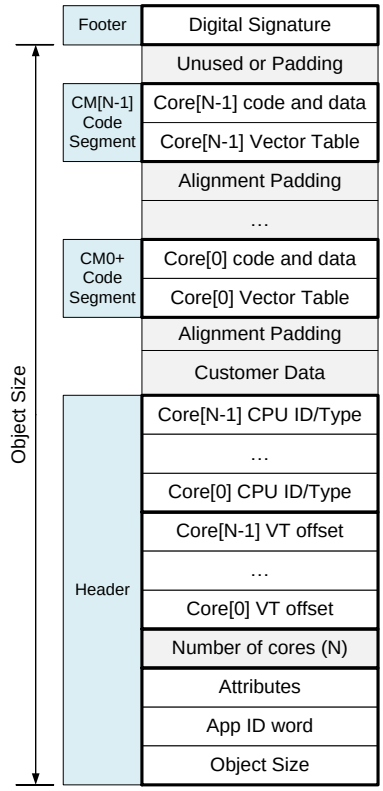


Table 3 lists details of header section in Cypress Standard Application Format. It defines the total size, the number of cores, the type of application, and the offset to each core application vector table.

Table 3. Header Details

Offset	Size	Item	Description
0x00	4 bytes	Object Size	A flash image size in bytes (application size)
0x04	4 bytes	Application ID/Version	This value identifies the type of the flash image. Bit 31 - 28: Always 0 Bit 27 - 24: Major version Bit 23 - 16: Minor version Bit 15 - 0: Application ID. For example, 0x0000 - User application 0x8001 - flash boot 0x8002 - Security library 0x8003 - Bootloader All other values - Reserved
0x08	4 bytes	Attribute	Reserved for future use
0x0C	4 bytes	Number of Cores(N)	Number of cores used by the application.
0x10 + (4*i)	4 bytes	Core(i) VT offset	Offset to the vector table from this address in Core(i) code segment

Offset	Size	Item	Description
0x10+(4*N)+(4*i)	4 bytes	Core(i) CPU ID and Core Index	The user assigned CPU ID and core index. Bit 31 - 20: CPU ID. This is the part of value from the CPUID [15:4] register in an Arm device. Bit 7 - 0: Core Index The core index is used to distinguish between multiple cores of the same type. For example, consider a system consisting of CM0+ and two CM7_0/CM7_1s. The CM0+ is identified by CPUID=0xC60 and Core Index=0. The first CM7_0/CM7_1 is identified by CPUID=0xC24 and Core Index=0. The second CM7_0/CM7_1 is identified by CPUID=0xC24 and Core Index=1.

To generate proper values for the application header, an instance of the following code can be included in the project.

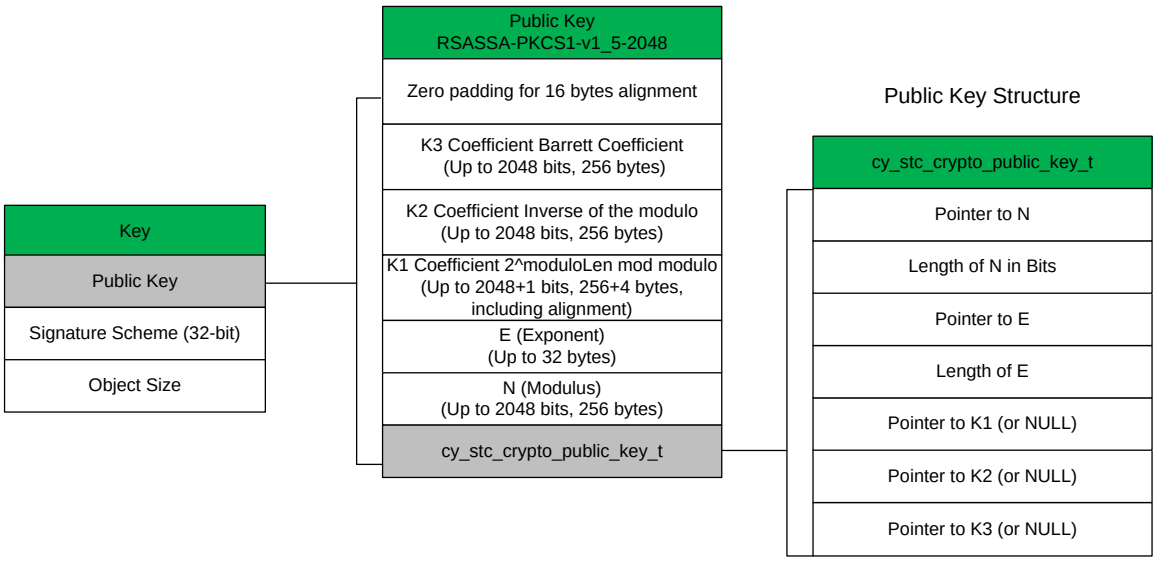
```
/** Secure Application header */
CY_SECTION(".cy_app_header") __USED static const cy_stc_si_appheader_t cy_si_appHeader =
{
    .objSize        = CY_M0PLUS_SI_SIZE,
    .appId          = (CY_SI_APP_VERSION | CY_SI_APP_ID_SECUREIMG),
    .appAttributes  = 0UL,                          /* Reserved */
    .numCores       = 1UL,                          /* Only CM0+ */
    .core0Vt        = CY_SI_VT_OFFSET,              /* CM0+ VT offset */
    .core0Id        = CY_SI_CPUID | CY_SI_CORE_IDX,  /* CM0+ core ID */
};

/** Secure Image Digital signature (Populated by cymcuelftool) */
CY_SECTION(".cy_app_signature") __USED CY_ALIGN(4)
static const uint8_t cy_si_appSignature[CY_SI_SECURE_DIGSIG_SIZE] = {0u};
```

8.3 Secure Boot RSA Public Key Format

The algorithm for encrypting is SHA-256 + RSASSA PKCS v1.5 algorithms as the scheme for signature verification. The RSA Key location is defined by the user. The user should provide a pointer to the key in the TOC2. The public key contents are checked by ROM boot HASH computation against any change.

The SFlash region stores the public key in a binary format. The modulus, exponent, and three coefficients are pre-calculated to speed up the validation. Figure 8 shows the format.

Figure 8. Public Key Format



The key is stored in three structures. The first structure “Key” is stored as an object that can easily be included in the SECURE_HASH calculation. The “Signature Scheme” defines the structure of the key. The “Object Size” contains the full size of the public key object, which contains the entire three structures. The second structure contains the individual pieces of the public key: coefficients (K1, K2, K3), exponent (E), and modulus (N). These values must be stored in a little-endian list of bytes. The third structure is a list of pointers to each piece of the public key, which is the format required for a call to the SROM firmware.

To generate proper values for the public key format, an instance of the following code must be included in the project.

More details of generating and using the private and public keys are discussed in Appendix A.

```
/** Public key in SFlash */
CY_SECTION(".cy_SFlash_public_key") __USED const cy_si_stc_public_key_t cy_publicKey =
{
    .objSize = sizeof(cy_si_stc_public_key_t),
    .signatureScheme = CY_SI_PUBLIC_KEY_RSA_2048,
    .publicKeyStruct =
    {
        .moduloAddr         = CY_SI_PUBLIC_KEY + offsetof(cy_si_stc_public_key_t, moduloData),
        .moduloSize         = CY_SI_PUBLIC_KEY_SIZEOF_BYTE * CY_SI_PUBLIC_KEY_MODULOLENGTH,
        .expAddr            = CY_SI_PUBLIC_KEY + offsetof(cy_si_stc_public_key_t, expData),
        .expSize            = CY_SI_PUBLIC_KEY_SIZEOF_BYTE * CY_SI_PUBLIC_KEY_EXPLENGTH,
        .barrettAddr        = CY_SI_PUBLIC_KEY + offsetof(cy_si_stc_public_key_t, barrettData),
        .inverseModuloAddr  = CY_SI_PUBLIC_KEY + offsetof(cy_si_stc_public_key_t, inverseModuloData),
        .rBarAddr           = CY_SI_PUBLIC_KEY + offsetof(cy_si_stc_public_key_t, rBarData),
    },
    .moduloData =
    { // N(Modulus)
    },
    .expData =
    { // E(Exponent)
    },
    .barrettData =
    { // K1
    },
    .inverseModuloData =
    { // K2
    },
    .rBarData =
    { // K3
    },
};
```

Appendix A Example of Creating Public and Private Keys

This project is provided with two files, *rsa_private.txt* and *rsa_public.txt*. These files contain a sample private and public key, which are provided only as placeholders to help you get your build system working properly. They should be replaced with your own files before going into production. This section explains how to generate a set of public and private keys, to format them to a 'C' format, and to update the source file with the new key.

A.1 Additional Tools Required

1. OpenSSL v1.0.2 or later
2. Python 3 (Required for one of the provided scripts that is used to format the public key.)

There are several ways to generate RSA private and public keys. In the following method, you need to install OpenSSL/Python on your computer. These source or binaries for OpenSSL can be downloaded from several sources on the internet.

A.2 Scripts

This project provides two scripts to convert the output from OpenSSL to a format compatible with C and the structures used by the Secure Image to store the public key. These scripts are available in the following path:

<user>\TVIIBE1M_RMA_si\04_Util\Scripts/Key <user>: Sample project stored folder

The batch script *rsa_keygen.bat* calls OpenSSL functions. Therefore, OpenSSL must be installed on your computer. The batch file creates a directory called "keys_generated". This creates two files containing the private and public keys generated with OpenSSL. These files are called "rsa_private.txt" for the private key, and "rsa_public.txt" for the public key.

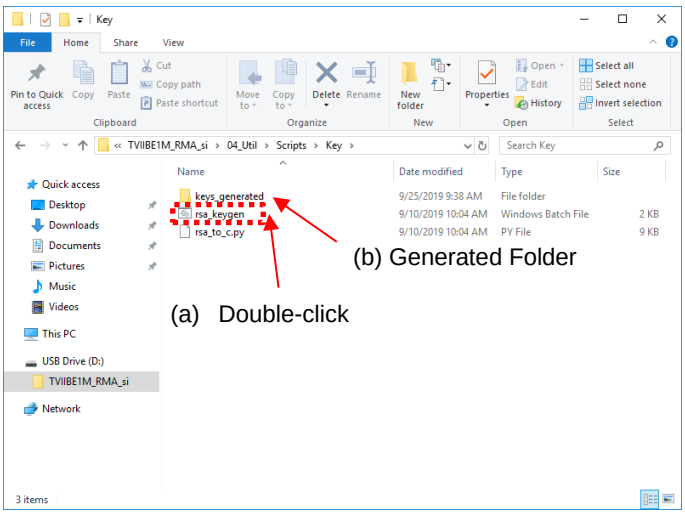
Next, the batch file will call the Python script, "rsa_to_c.py". This script formats the data to be compatible with C and the Cypress public key format from the generated public key file. The output is placed in the file *rsa_to_c_generated.txt* in the *keys_generated* directory along with the public and private key files.

A.3 Running the Scripts

The batch file *rsa_keygen.bat* can be called from a command line interface or by double-clicking (a) it in a Windows 7 or 10 environment. After the script runs, verify that the following files have been generated in the *keys_generated* folder (b).

- rsa_private.txt (Private RSA key)
- rsa_public.txt (Public RSA key)
- rsa_to_c_generated.txt (Public key in C format)

Figure 9. Creating Key



An example of what the private key file should look like:

```
-----BEGIN RSA PRIVATE KEY-----
MIIEogIBAAKCAQEAs1LSkjDJbsnXvHdMbMYB4SMCN2QbMRwJmYytIVpc7kylFZN0
iK+uj1gB3UZzQshY6IjrAjCJy7l9xr8arzWu/waYntSshQmts4Bdd7UWefZtNjou
IRK6+N+geefXXEJFWzZd0/5Bdur+Z6gI8xUuc0BrgpBAEtVDf+/uBnPmPD3Eh7tu
60NbqkB22P9bwY0oyjKaGZUU0D0b8zCPwmp+RXHmLcK+D7q8CwotUkmYxpX/8PaV
ho6jwbCggI7FSWAF2m0KrtB1mkiAlKEavM+7mpsysYWApOghULrlqEy9svXAYDch
jaOmD1c7P2WGYxgmR/9acj1cMisemBCcMQmqOQIDAQABAoIBAG3OopjYfRB1UsDa
gJb+7Pmy2VGX4DrDQ3GePgn1boCRuGks4SguBANMzd90m3HigubpqJavqEFy9Xz1
Wt69TcpmfKCN9JKGqbIIQbPi1L+cUaXqHIMuGAX70iWz/SFGh3fiI7SDtgyeu2g3
8jr/1nDGms7ZjBK8Otgl0Khs4igyV8BXN+6xcoc5mzUW1T52vtsJFl1NpTN9IthD
Rm5WAb729mZNnWEixOHQ1BJZxJHFtUCZzK2O15rTMriWtgHo0teF/QHRpZzWxYhj
p4poIUHep7ZMQOAp5mVxWoIyUa1et3vVMfReVFBdDysi76lFvRsA7gTriJbAbEhX
hpuA3YECgYEA41EA6I98VTm07w6KyujmdIGwI4zLvRy25DSt8LaBNntUAFBJdBJB
/WGoPMfXMcj5d42wkpToO6FHqxqDZK+zNlYSCUAd7FeiRRlruRV7h1vFrAZt83el
UfuBLSmfDMUiagXE3I5qTm1Ccl/fk+v2xqz08BdNLHzEsRUgZWyii+kCgYEAyfOB
eYfayufO2K4MhTphEwBrcDaHM2Wbkqgcx+AT3ejFMA1ujHMDHSh0zaigodjl3ekY
FiYvXx1zi19wdpkOrjP+GMzhei70X8IcciszMB5ACmqWMDymGN58gP4616vbTkYa
/e4bZpta8E66xHbaf8iTFNGXrerdA+NAV5zwSdECgYBEbU3os8ipnwVoMOh2zEUT
LTbofKmX89zaNUFnBIlA7T8MGR8DYbubmpoaMc4Fnon6Axz0Av8ldBcu2nxTAtDf
umkTf8mCK58dTdmvjVfyFx4Z+pV+hLWvSBZKkgzeuzjw0WvBON2nXhxycYnL7WwL
wmS5IUqJ6cULyWBHJ7yGgQKBgHhosoLmiIJAUikJJTfuReDReD2QlW4EoAyCJZ7a
sJ234pIy//3HuUySSYoxh4zYSL3V8+GI5Oe+JJ0ty107BvDA2TiqN6nlAxlxIArG
TR/ceWx8fo3GK3ZaeTtj2Wur8Pcrf351kGrOKBttpZfsEXzs9x0LlndAuIRP45YZ
YX9xAoGARcsEcXuU9Mm7fprdKUjlxp1Tc/Ge+f3zYMCf9Arw+VnENUdUqMfeO4Z8
tjMd6eFC6/eaGpgOkhv85T22bi8xHNcQaM768nk4jb8sok/g+JG3k1+X0AulRTCY
Lbn8U9D4rYaRVvHeWKHzXxROlImqoJ7qe/T1MdltDJNR5Sy8Tm8=
-----END RSA PRIVATE KEY-----
```

An example of what the public key file should look like:

```
-----BEGIN PUBLIC KEY-----
MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAs1LSkjDJbsnXvHdMbMYB
4SMCN2QbMRwJmYytIVpc7kylFZN0iK+uj1gB3UZzQshY6IjrAjCJy7l9xr8arzWu
/waYntSshQmts4Bdd7UWefZtNjouIRK6+N+geefXXEJFWzZd0/5Bdur+Z6gI8xUu
c0BrgpBAEtVDf+/uBnPmPD3Eh7tu60NbqkB22P9bwY0oyjKaGZUU0D0b8zCPwmp+
RXHmLcK+D7q8CwotUkmYxpX/8PaVho6jwbCggI7FSWAF2m0KrtB1mkiAlKEavM+7
mpsysYWApOghULrlqEy9svXAYDchjaOmD1c7P2WGYxgmR/9acj1cMisemBCcMQmq
OQIDAQAB
-----END PUBLIC KEY-----
```

A.4 Installing the Public Key

The final step to updating the public key in the Secure Image is to copy the code in the generated *rsa_to_c_generated.txt* file to the *main_cm0plus.c* source file, which is part of the Secure Image project. An example of this file after being updated is shown below. The replacement code from the generated key data is shown below in green.

```
/** Public key in SFlash */
CY_SECTION(".cy_SFlash_public_key") __USED const cy_si_stc_public_key_t cy_publicKey =
{
    .objSize = sizeof(cy_si_stc_public_key_t),
    .signatureScheme = CY_SI_PUBLIC_KEY_RSA_2048,
    .publicKeyStruct =
    {
        .moduloAddr         = CY_SI_PUBLIC_KEY + offsetof(cy_si_stc_public_key_t, moduloData),
        .moduloSize         = CY_SI_PUBLIC_KEY_SIZEOF_BYTE * CY_SI_PUBLIC_KEY_MODULOLENGTH,
        .expAddr            = CY_SI_PUBLIC_KEY + offsetof(cy_si_stc_public_key_t, expData),
        .expSize            = CY_SI_PUBLIC_KEY_SIZEOF_BYTE * CY_SI_PUBLIC_KEY_EXPLENGTH,
        .barrettAddr        = CY_SI_PUBLIC_KEY + offsetof(cy_si_stc_public_key_t, barrettData),
        .inverseModuloAddr  = CY_SI_PUBLIC_KEY + offsetof(cy_si_stc_public_key_t, inverseModuloData),
        .rBarAddr           = CY_SI_PUBLIC_KEY + offsetof(cy_si_stc_public_key_t, rBarData),
    },
    .moduloData =
    {
      0x39u, 0xAAu, 0x09u, 0x31u, 0x9Cu, 0x10u, 0x98u, 0x1Eu,
      0x2Bu, 0x32u, 0x5Cu, 0x3Du, 0x72u, 0x5Au, 0xFFu, 0x47u,
```

```
  0x26u, 0x18u, 0x63u, 0x86u, 0x65u, 0x3Fu, 0x3Bu, 0x57u,
  0x0Fu, 0xA6u, 0xA3u, 0x8Du, 0x21u, 0x37u, 0x60u, 0xC0u,
  0xF5u, 0xB2u, 0xBDu, 0x4Cu, 0xA8u, 0xE5u, 0xBAu, 0x50u,
  0x21u, 0xE8u, 0xA4u, 0x80u, 0x85u, 0xB1u, 0x32u, 0x9Bu,
  0x9Au, 0xBBu, 0xCFu, 0xBCu, 0x1Au, 0xA1u, 0x94u, 0x80u,
  0x48u, 0x9Au, 0x75u, 0xD0u, 0xAEu, 0x0Au, 0x6Du, 0xDAu,
  0x05u, 0x60u, 0x49u, 0xC5u, 0x8Eu, 0x80u, 0xA0u, 0xB0u,
  0xC1u, 0xA3u, 0x8Eu, 0x86u, 0x95u, 0xF6u, 0xF0u, 0xFFu,
  0x95u, 0xC6u, 0x98u, 0x49u, 0x52u, 0x2Du, 0x0Au, 0x0Bu,
  0xBCu, 0xBAu, 0x0Fu, 0xBEu, 0xC2u, 0x2Du, 0xE6u, 0x71u,
  0x45u, 0x7Eu, 0x6Au, 0xC2u, 0x8Fu, 0x30u, 0xF3u, 0x1Bu,
  0x3Du, 0xD0u, 0x14u, 0x95u, 0x19u, 0x9Au, 0x32u, 0xCAu,
  0x28u, 0x8Du, 0xC1u, 0x5Bu, 0xFFu, 0xD8u, 0x76u, 0x40u,
  0xAAu, 0x5Bu, 0x43u, 0xEBu, 0x6Eu, 0xBBu, 0x87u, 0xC4u,
  0x3Du, 0x3Cu, 0xE6u, 0x73u, 0x06u, 0xEEu, 0xEFu, 0x7Fu,
  0x43u, 0xD5u, 0x12u, 0x40u, 0x90u, 0x82u, 0x6Bu, 0x40u,
  0x73u, 0x2Eu, 0x15u, 0xF3u, 0x08u, 0xA8u, 0x67u, 0xFEu,
  0xEAu, 0x76u, 0x41u, 0xFEu, 0xD3u, 0x5Du, 0x36u, 0x5Bu,
  0x45u, 0x42u, 0x5Cu, 0xD7u, 0xE7u, 0x79u, 0xA0u, 0xDFu,
  0xF8u, 0xBAu, 0x12u, 0x21u, 0x2Eu, 0x3Au, 0x36u, 0x6Du,
  0xF6u, 0x79u, 0x16u, 0xB5u, 0x77u, 0x5Du, 0x80u, 0xB3u,
  0xADu, 0x09u, 0x85u, 0xACu, 0xD4u, 0x9Eu, 0x98u, 0x06u,
  0xFFu, 0xAEu, 0x35u, 0xAFu, 0x1Au, 0xBFu, 0xC6u, 0x7Du,
  0xB9u, 0xCBu, 0x89u, 0x30u, 0x02u, 0xEBu, 0x88u, 0xE8u,
  0x58u, 0xC8u, 0x42u, 0x73u, 0x46u, 0xDDu, 0x01u, 0x58u,
  0x8Fu, 0xAEu, 0xAFu, 0x88u, 0x74u, 0x93u, 0x15u, 0xA5u,
  0x4Cu, 0xEEu, 0x5Cu, 0x5Au, 0x21u, 0xADu, 0x8Cu, 0x99u,
  0x09u, 0x1Cu, 0x31u, 0x1Bu, 0x64u, 0x37u, 0x02u, 0x23u,
  0xE1u, 0x01u, 0xC6u, 0x6Cu, 0x4Cu, 0x77u, 0xBCu, 0xD7u,
  0xC9u, 0x6Eu, 0xC9u, 0x30u, 0x92u, 0xD2u, 0x52u, 0xB3u,
},
.expData =
{
  0x01u, 0x00u, 0x01u, 0x00u,
},
.barrettData =
{
  0xA2u, 0x78u, 0x5Cu, 0x68u, 0x49u, 0xBBu, 0x85u, 0xD6u,
  0xF0u, 0x36u, 0xE3u, 0xAAu, 0xF7u, 0x33u, 0x48u, 0x40u,
  0xC2u, 0xE2u, 0x75u, 0x03u, 0x7Eu, 0x18u, 0xCAu, 0x0Bu,
  0x21u, 0xF3u, 0xDFu, 0x70u, 0xF4u, 0x73u, 0xBCu, 0x4Bu,
  0xA2u, 0xFDu, 0x98u, 0x3Cu, 0x71u, 0x20u, 0xD3u, 0xECu,
  0x57u, 0xC4u, 0xFEu, 0xE5u, 0xBBu, 0x38u, 0xEEu, 0x0Bu,
  0x38u, 0x25u, 0xA5u, 0x0Au, 0xABu, 0xF5u, 0x88u, 0xE5u,
  0x8Eu, 0x98u, 0xA7u, 0xA6u, 0x6Du, 0x2Fu, 0x12u, 0x40u,
  0xC3u, 0x2Du, 0xD5u, 0x34u, 0x15u, 0x7Du, 0x6Au, 0x18u,
  0xE8u, 0x64u, 0x3Au, 0x47u, 0x1Eu, 0xAFu, 0x0Cu, 0x8Eu,
  0x75u, 0xE0u, 0x39u, 0x2Cu, 0x09u, 0x8Cu, 0xE0u, 0x96u,
  0x6Du, 0xD4u, 0xB4u, 0x9Bu, 0x77u, 0xF0u, 0xA8u, 0xDAu,
  0x7Cu, 0x60u, 0x09u, 0xF0u, 0x82u, 0xACu, 0x68u, 0x14u,
  0x46u, 0xEEu, 0x1Du, 0xF7u, 0xCCu, 0x45u, 0xE8u, 0xCAu,
  0x83u, 0x5Au, 0x19u, 0x74u, 0x1Bu, 0xEFu, 0xBAu, 0x98u,
  0x4Bu, 0xC7u, 0x20u, 0x97u, 0x15u, 0xC8u, 0x8Bu, 0x17u,
  0x09u, 0x06u, 0xB3u, 0x6Fu, 0x85u, 0x7Du, 0xC5u, 0x72u,
  0xDCu, 0xD3u, 0x8Du, 0x14u, 0x12u, 0x8Bu, 0x6Cu, 0x81u,
  0x33u, 0x6Fu, 0x57u, 0xF2u, 0x3Bu, 0x1Fu, 0x66u, 0x1Cu,
  0xF9u, 0x3Au, 0xE3u, 0xE3u, 0x3Eu, 0x1Du, 0x86u, 0xDCu,
  0xDCu, 0x85u, 0x29u, 0xD2u, 0x83u, 0x35u, 0x83u, 0x1Du,
  0x44u, 0x51u, 0xD3u, 0x68u, 0x74u, 0x6Au, 0xBFu, 0xAEu,
  0x3Eu, 0xCDu, 0x2Bu, 0xC6u, 0x7Fu, 0xDDu, 0xB5u, 0xB8u,
  0x3Eu, 0x6Au, 0xEFu, 0x72u, 0x14u, 0xE9u, 0x56u, 0xBEu,
  0xD0u, 0xD2u, 0xA0u, 0xA5u, 0x0Du, 0x68u, 0xA4u, 0x4Du,
  0x76u, 0x7Au, 0x1Fu, 0xDFu, 0xD8u, 0x19u, 0x84u, 0x4Cu,
  0x5Eu, 0xE4u, 0x5Fu, 0x1Au, 0xD7u, 0x7Bu, 0x79u, 0xCEu,
  0xF9u, 0xFFu, 0x2Fu, 0x0Au, 0xFFu, 0xC5u, 0x3Au, 0xA8u,
  0xFAu, 0x62u, 0xC5u, 0xDEu, 0x75u, 0xE7u, 0x22u, 0x01u,
  0x4Du, 0x48u, 0x15u, 0x76u, 0x79u, 0x35u, 0x25u, 0x9Du,
  0x33u, 0x0Fu, 0xFAu, 0xA5u, 0xE7u, 0x41u, 0xEDu, 0x06u,
  0xD0u, 0x83u, 0x4Bu, 0xC4u, 0xA4u, 0x5Du, 0x76u, 0x6Du,
  0x01u, 0x00u, 0x00u, 0x00u,
},
.inverseModuloData =
{
  0xF7u, 0xDBu, 0x7Eu, 0xBBu, 0x40u, 0x73u, 0x6Eu, 0x72u,
  0xEFu, 0xA6u, 0x8Au, 0x7Fu, 0x8Au, 0x28u, 0x8Du, 0xB5u,
  0x35u, 0x2Fu, 0xD7u, 0x6Cu, 0x67u, 0x0Au, 0xBAu, 0xE3u,
  0x0Cu, 0xFEu, 0x8Fu, 0xDBu, 0x86u, 0xA7u, 0x3Cu, 0xC4u,
  0xACu, 0x26u, 0xF9u, 0x57u, 0x82u, 0xCAu, 0x66u, 0xC9u,
  0x76u, 0x9Fu, 0x3Bu, 0x36u, 0x38u, 0x14u, 0x72u, 0xF2u,
```

```
    0x28u, 0xFCu, 0xBDu, 0x2Eu, 0xFDu, 0x65u, 0x89u, 0x35u,
    0x78u, 0x7Du, 0x99u, 0x07u, 0x1Au, 0x53u, 0xC8u, 0x3Eu,
    0x51u, 0xD3u, 0xF2u, 0xFDu, 0xCEu, 0x92u, 0x8Fu, 0x10u,
    0xD2u, 0x27u, 0xC7u, 0xCCu, 0x0Fu, 0xF4u, 0xC9u, 0xAEu,
    0xCEu, 0x50u, 0x68u, 0x8Cu, 0x76u, 0xE9u, 0x91u, 0xD9u,
    0x42u, 0x55u, 0x1Fu, 0x25u, 0x04u, 0xB1u, 0xBDu, 0xABu,
    0xA1u, 0x16u, 0xBCu, 0xD7u, 0x2Cu, 0x8Bu, 0x55u, 0xC2u,
    0x02u, 0x96u, 0x04u, 0x44u, 0xB4u, 0x71u, 0x88u, 0xF9u,
    0x79u, 0xD0u, 0xF0u, 0x2Du, 0x58u, 0xF9u, 0x93u, 0xD5u,
    0x91u, 0x24u, 0xB8u, 0x2Bu, 0xA9u, 0x3Eu, 0x6Au, 0xE3u,
    0x07u, 0x44u, 0xDCu, 0xD5u, 0x8Du, 0xB1u, 0xA3u, 0xC3u,
    0x09u, 0x57u, 0xC5u, 0x9Au, 0xAEu, 0x93u, 0x0Cu, 0xEEu,
    0x29u, 0xEAu, 0x03u, 0x41u, 0xD0u, 0xE6u, 0xA1u, 0xFFu,
    0x65u, 0x02u, 0x17u, 0x7Eu, 0x31u, 0x3Cu, 0x00u, 0x4Cu,
    0xA9u, 0x32u, 0xF3u, 0xC6u, 0x8Du, 0xA9u, 0x33u, 0xDBu,
    0x62u, 0x23u, 0x4Eu, 0xE3u, 0x1Au, 0xEAu, 0x97u, 0x60u,
    0xA8u, 0x34u, 0xE3u, 0x3Bu, 0x96u, 0xBCu, 0xE5u, 0x2Fu,
    0xC2u, 0x66u, 0x40u, 0xE6u, 0xFFu, 0x92u, 0x84u, 0xF6u,
    0x38u, 0xB7u, 0x59u, 0x81u, 0x96u, 0xEFu, 0x1Fu, 0xD9u,
    0xA9u, 0x20u, 0x8Bu, 0xB2u, 0x77u, 0x49u, 0x0Fu, 0xA9u,
    0x0Fu, 0x7Fu, 0x60u, 0xD4u, 0x6Bu, 0xBAu, 0xC6u, 0x73u,
    0xA2u, 0x25u, 0x44u, 0xA2u, 0xEAu, 0x91u, 0xDBu, 0xA3u,
    0xC2u, 0x8Cu, 0x27u, 0x38u, 0xFCu, 0xEAu, 0xFEu, 0x00u,
    0x6Du, 0x93u, 0xB6u, 0x0Du, 0xF8u, 0x74u, 0xFDu, 0x14u,
    0xC7u, 0xD5u, 0xE7u, 0x7Du, 0x32u, 0x08u, 0x52u, 0x8Du,
    0xACu, 0x66u, 0x03u, 0x4Fu, 0xA9u, 0x33u, 0xA0u, 0x7Bu,
  },
  .rBarData =
  {
    0xC7u, 0x55u, 0xF6u, 0xCEu, 0x63u, 0xEFu, 0x67u, 0xE1u,
    0xD4u, 0xCDu, 0xA3u, 0xC2u, 0x8Du, 0xA5u, 0x00u, 0xB8u,
    0xD9u, 0xE7u, 0x9Cu, 0x79u, 0x9Au, 0xC0u, 0xC4u, 0xA8u,
    0xF0u, 0x59u, 0x5Cu, 0x72u, 0xDEu, 0xC8u, 0x9Fu, 0x3Fu,
    0x0Au, 0x4Du, 0x42u, 0xB3u, 0x57u, 0x1Au, 0x45u, 0xAFu,
    0xDEu, 0x17u, 0x5Bu, 0x7Fu, 0x7Au, 0x4Eu, 0xCDu, 0x64u,
    0x65u, 0x44u, 0x30u, 0x43u, 0xE5u, 0x5Eu, 0x6Bu, 0x7Fu,
    0xB7u, 0x65u, 0x8Au, 0x2Fu, 0x51u, 0xF5u, 0x92u, 0x25u,
    0xFAu, 0x9Fu, 0xB6u, 0x3Au, 0x71u, 0x7Fu, 0x5Fu, 0x4Fu,
    0x3Eu, 0x5Cu, 0x71u, 0x79u, 0x6Au, 0x09u, 0x0Fu, 0x00u,
    0x6Au, 0x39u, 0x67u, 0xB6u, 0xADu, 0xD2u, 0xF5u, 0xF4u,
    0x43u, 0x45u, 0xF0u, 0x41u, 0x3Du, 0xD2u, 0x19u, 0x8Eu,
    0xBAu, 0x81u, 0x95u, 0x3Du, 0x70u, 0xCFu, 0x0Cu, 0xE4u,
    0xC2u, 0x2Fu, 0xEBu, 0x6Au, 0xE6u, 0x65u, 0xCDu, 0x35u,
    0xD7u, 0x72u, 0x3Eu, 0xA4u, 0x00u, 0x27u, 0x89u, 0xBFu,
    0x55u, 0xA4u, 0xBCu, 0x14u, 0x91u, 0x44u, 0x78u, 0x3Bu,
    0xC2u, 0xC3u, 0x19u, 0x8Cu, 0xF9u, 0x11u, 0x10u, 0x80u,
    0xBCu, 0x2Au, 0xEDu, 0xBFu, 0x6Fu, 0x7Du, 0x94u, 0xBFu,
    0x8Cu, 0xD1u, 0xEAu, 0x0Cu, 0xF7u, 0x57u, 0x98u, 0x01u,
    0x15u, 0x89u, 0xBEu, 0x01u, 0x2Cu, 0xA2u, 0xC9u, 0xA4u,
    0xBAu, 0xBDu, 0xA3u, 0x28u, 0x18u, 0x86u, 0x5Fu, 0x20u,
    0x07u, 0x45u, 0xEDu, 0xDEu, 0xD1u, 0xC5u, 0xC9u, 0x92u,
    0x09u, 0x86u, 0xE9u, 0x4Au, 0x88u, 0xA2u, 0x7Fu, 0x4Cu,
    0x52u, 0xF6u, 0x7Au, 0x53u, 0x2Bu, 0x61u, 0x67u, 0xF9u,
    0x00u, 0x51u, 0xCAu, 0x50u, 0xE5u, 0x40u, 0x39u, 0x82u,
    0x46u, 0x34u, 0x76u, 0xCFu, 0xFDu, 0x14u, 0x77u, 0x17u,
    0xA7u, 0x37u, 0xBDu, 0x8Cu, 0xB9u, 0x22u, 0xFEu, 0xA7u,
    0x70u, 0x51u, 0x50u, 0x77u, 0x8Bu, 0x6Cu, 0xEAu, 0x5Au,
    0xB3u, 0x11u, 0xA3u, 0xA5u, 0xDEu, 0x52u, 0x73u, 0x66u,
    0xF6u, 0xE3u, 0xCEu, 0xE4u, 0x9Bu, 0xC8u, 0xFDu, 0xDCu,
    0x1Eu, 0xFEu, 0x39u, 0x93u, 0xB3u, 0x88u, 0x43u, 0x28u,
    0x36u, 0x91u, 0x36u, 0xCFu, 0x6Du, 0x2Du, 0xADu, 0x4Cu,
  },
};
```

The `Cy_FB_Isvalidkey` can check whether the public key structure is valid. See the “Flash Boot” chapter in Architecture TRM for more details.

Appendix B Creating a Secure Image

This project provides a *Secure_Complete_SREC_to_CYP.bat* script for signing the secure code. These scripts are available in the following path:

<user>\TVIIBE1M_RMA_si\04_Util\Scripts *<user>: Sample project folder*

This script adds the digital signature to the CM0+ ELF file and generates a CM0+ SREC file in the *CYP_Scripts* folder. Also, it copies CM4 ELF/SREC files into the *CYP_Scripts* folder.

This script calls the *cymcuelftool.exe* tool. This tool generates and adds digital signature to CM0+ ELF file. This tool is stored in *<user>\TVIIBE1M_RMA_si\04_Util\Tools*.

This script generates a digital signature using the private key, and adds the digital signature to the secure code. The generated digital signature is then decrypted using the public key stored in SFlash and is validated by flash boot.

The following is an example of using the *cymcuelftool.exe* tool.

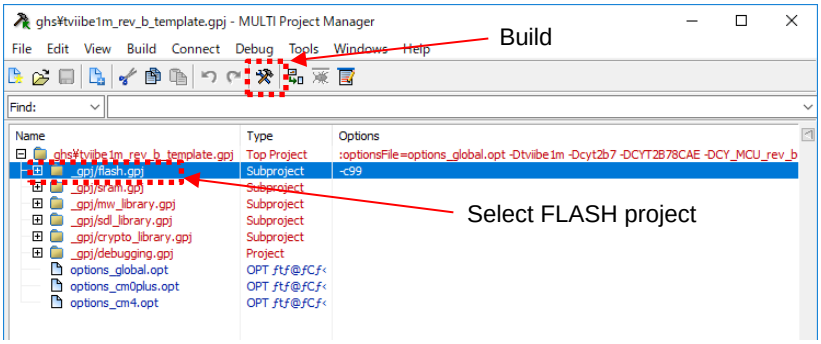
```
> cymcuelftool.exe –sign [in.elf] [HASH ALGORITHM] –encrypt [ENC_ALGORITHM] –key
[PrivateKey] –output [out.elf]
```

You can also add a digital signature to a CM4 ELF file by selecting CM4 ELF file to in.elf.

B.1 Building and Programming

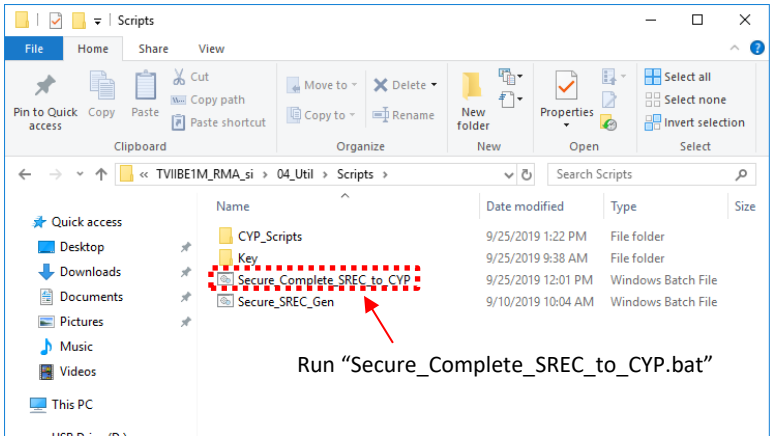
1. Build a flash project using GHS Multi.

Figure 10. Build Flash Project



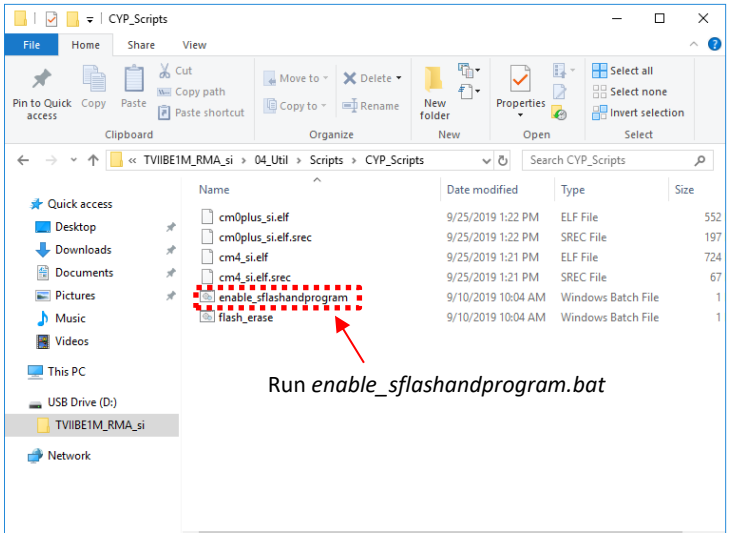
2. Run the *Secure_Complete_SREC_to_CYP.bat* script.
The CM0+ SREC file (*cm0plus_si.elf.srec*) with the digital signature added and the CM4 SREC file (*cm4_si.elf.srec*) are stored to the *CYP_Scripts* folder.

Figure 11. Run *Secure_Complete_SREC_to_CYP.bat*



3. Connect MiniProg4 to the target board.
4. Run *enable_sflashandprogram.bat*.
   The CM0+ SREC file and CM4 SREC file are programmed into the MCU device.

Figure 12. Run *enable_sflashandprogram.bat*



Appendix C Requirements for Generating a Digital Signature

The *cymcuelftool.exe* tool requires the following symbols/sections for digital signature generation.

a) ".cy_app_signature" section:
This section specifies the location the digital signature is written to:

```
/** Secure Image Digital signature (Populated by cymcuelftool) */
CY_SECTION(".cy_app_signature") __USED CY_ALIGN(4)
static const uint8_t cy_si_appSignature[CY_SI_SECURE_DIGSIG_SIZE] = {0u};
```

b) "__cy_app_verify_start" and "__cy_app_verify_length" symbols:
This symbol defines the first address and size of the memory area whose digital signature is calculated.

```
__cy_app_verify_start = 0x10000000;
__cy_app_verify_length = 0xFF00;
```

In the example above, the digital signature calculation area is from 0x10000000 to 0x1000FF00.

Note that in the digital signature generated by this script, blanks in the calculation area expect "0". See the Appendix I for how to generate SREC file with "0".

Appendix D Authentication of the Main User Application

The Main User Application is authenticated by the Secure Image. For CoT, the main user application must be authenticated before the main CPU (CM4 or CM7) is activated.

Traveo II supports the `Cy_FB_VerifyApplication` function to authenticate user applications. This function is included in flash boot and can be executed from the user code. This function can be used to authenticate the other code with RSASSA-PKCS1-v1.5. See the “Flash Boot” chapter in Architecture TRM for more details.

In this example, a digital signature is generated using the same method as the first user application, and authentication is performed using the public key in SFlash.

```
typedef bool (*pFB_VerifyApp)(uint32_t, uint32_t, uint32_t, uint32_t);


int main(void)
{
    bool isAppValid;
    uint32_t publicKeyAddr;
    uint32_t appDataAddr;
    uint32_t appDataLength;
    uint32_t appSignatureAddr;
    uint32_t toc2Addr;

    pFB_VerifyApp VerifyApplication = (pFB_VerifyApp)*(uint32_t*)(0x17002040);

    SystemInit();

    /* Enable global interrupts. */
    __enable_irq();

    /* Run RSA */
    appDataAddr       = CY_SI_SECURE_FLASH_BEGIN_CM4;
    appDataLength     = CY_M4_SI_SIZE;
    appSignatureAddr  = CY_SI_SIGNATURE_ADDR;
    publicKeyAddr     = CY_SI_PUBLIC_KEY + CY_FB_PBKEY_STRUCT_OFFSET;

    isAppValid = VerifyApplication (appDataAddr, appDataLength, appSignatureAddr, publicKeyAddr);

    /* RSA Verify Error Handling Here */
    while(isAppValid == false);

    /* Enable CM4.  */
    Cy_SysEnableApplCore(CY_CORTEX_M4_APPL_ADDR);

    while(1)
    {
    }
}
```

This function requires the following parameters:

Table 4. Cy_FB_VerifyApplication Function Parameters

Parameter	Description
`appDataAddr`	The start address of a data image to be authenticated
`appDataLength`	The length of the data image
`appSignatureAddr`	The start address of the signature for the data image
`publicKeyAddr`	The address of a public key structure

Appendix E Transition to RMA Stage

The customer transitions the part to RMA lifecycle stage when the customer wants Cypress to perform failure analysis on the part.

A device cannot transition to RMA stage without knowing the device unique ID and the customer's private key that is paired with a public key stored in SFlash. The customer must implement at least two special commands that can be sent from outside the device via UART, SPI, I^2C, CAN, LIN, and so forth. The first command is to read the internal unique device ID, and the other is to invoke the transition to RMA.

The customer must generate a certificate to transition the part with a specific Unique ID to the RMA lifecycle stage. After the part has been transitioned to RMA, the customer must provide Cypress another certificate that is signed by the using their private key for failure analysis on the part.

Do the following to transition the device to the RMA stage.

1. Erase all sensitive or proprietary code stored in the device. This may be performed with a special command or with a special code image described earlier. Erase the flash at least multiple times to ensure that there is no way to detect any residual code. The public key stored in SFlash cannot be erased because it is used to transition to RMA lifecycle stage, and to open the RMA device later by Cypress.
2. Read the device unique ID stored in the devices SFlash. This can be done by invoking a system call 0x1F (Read Unique ID), then sending the ID out via the communication interface.
3. Use the unique ID and the customer's private key that is paired with the public key stored internally in SFlash to generate a certificate. Table 5 shows the format of the certificate.
4. Send a command to the device that includes this certificate. The user must implement code to accept this certificate to invoke the transition to RMA system call (0x28) and pass the certificate as its parameter.
5. After the device is reset or power cycled, it will wait in a state for a single command from the debug port to open RMA (System call 0x29).

Table 5. RMA Certificate Format

Object	Bytes
Object Size	4 bytes
Command ID	4 bytes TransitionToRMA: 0x120028F0 OpenRMA: 0x120029F0
Unique ID	11 bytes
Zero Padding	1 byte
Digital Signature	256 bytes

Note that the certificates and digital signatures for TransitionToRMA and OpenRMA are different.

E.1 Generate Certificate

This section describes how to generate certificate in Linux using OpenSSL.

1. Use the `ReadUniqueID` API (System call 0x1F) to read the Unique ID in SFlash. The following is an example of the read result of the read unique ID.

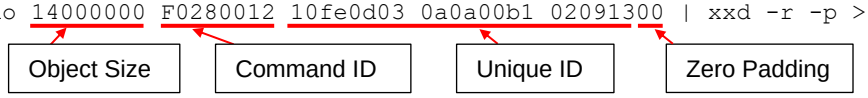
   0xa00dfe10: First byte is the system call status (0xa0: indicating success / 0xf0: indicating fail)
   Others: Unique ID_0

   0x000a0a03: Unique ID_1

   0x130902b1: Unique ID_2

2. Enter the following commands from the Linux environment. The following is an example of TransitionToRMA certificate and digital signature generation.

```
echo 14000000 F0280012 10fe0d03 0a0a00b1 02091300 | xxd -r -p > _data.bin
```




```
openssl dgst -sha256 -sign rsa_private.txt _data.bin > _signature.bin

cat _signature.bin | xxd -p -c 64 > _tmp1.hex

openssl rsautl -inkey rsa_private.txt -encrypt -raw -in _signature.bin | xxd -p -c64

echo 00000028 > _tmp2.hex

echo 14000000 F0280012 10fe0d03 0a0a00b1 02091300 >> _tmp2.hex

cat _tmp1.hex >> _tmp2.hex

sed -r "s/\s*(\w{2})(\w{2})(\w{2})(\w{2})/0x\4\3\2\1\n/g" _tmp2.hex | sed -r "/^$/d"
> _output_transtorma.hex

rm _data.bin _tmp1.hex _tmp2.hex
```

_output_transtorma.hex file is shown below:

```
0x28000000
0x00000014
0x120028F0
0x030dfe10
0xb1000a0a
0x00130902
0x788a21b0
0x0f71d29c
0x20a4f3c2
0x57852970
0xf46cdda9
0xc9cc67cd
    :
0x077cad56
0x6005452c
0xfd2e0a7f
0x6d78a91d
0xfca46879
0x6225e2a6
0x317ba835
0xc02d7263
0x8fcc5189
0xa5a57fb8
0x73a073d8
0x1126a6f7
```

Certificate

Digital Signature (256 bytes)

Appendix F Configure Application Protection

This project provides a means of configuring Application Protection. This section describes how you can configure and add SWPU, which is Application Protection for the customer system. SWPU consists of FWPU, ERPU, and EWPU. Application Protection can be configured with up to 16 entries for FWPU and up to 4 entries for ERPU and EWPU.

Each SWPU has master/slave structure, that is, it protects the protection structure by protection structure. The slave attribute indicates access attribute to resources and the master protection attribute indicates access attribute to slave. Therefore, changes in the SWPU slave attribute requires master-defined attributes.

Application protection consists of the following elements. See the “Protection Unit” chapter in Architecture TRM for more details.

- PU_OBJECT_SIZE: Number of configured elements. (4 bytes)
- N_FWPU: Number of FWPU objects. FWPU has up to 16 regions. (4 bytes)
- FWPUx_SL_ADDR: Configures the FWPUx base address. (4 bytes)
- FWPUx_SL_SIZE: Configures the FWPUx region size and FWPUx enable. (4 bytes)
- FWPUx_SL_ATT: Configures the FWPUx slave attribute. (4 bytes)
- FWPUx_MS_ATT: Configures the FWPUx master attribute. (4 bytes)
- N_ERPU: Number of ERPU objects. ERPU has up to 4 regions. (4 bytes)
- ERPUy_SL_OFFSET: Configures the ERPUy base address offset. (4 bytes)
- ERPUy_SL_SIZE: Configures the ERPUy region size and ERPUy enable. (4 bytes)
- ERPUy_SL_ATT: Configures the ERPUy slave attribute. (4 bytes)
- ERPUy_MS_ATT: Configures the ERPUy master attribute. (4 bytes)
- N_EWPU: Number of EWPU objects. EWPU has up to 4 regions. (4 bytes)
- EWPUy_SL_OFFSET: Configures the EWPUy base address offset. (4 bytes)
- EWPUy_SL_SIZE: Configures the EWPUy region size and ERPUy enable. (4 bytes)
- EWPUy_SL_ATT: Configures the EWPUy slave attribute. (4 bytes)
- EWPUy_MS_ATT: Configures the EWPUy master attribute. (4 bytes)

Suffix “x” indicates 0 to 15, and suffix “y” indicates 0 to 3.

F.1 Configuration

The following shows the steps to configure application protection. This section address must match the "Address of Application Protection (offset = 0x108)" in TOC2. To ensure security, do not change the default value (0x17007600).

1. Set the number of regions for each SWPU.

```
// deification of application protection
#define N_FWPU                (0UL)  /**< Number of flash write protection Max 16 */
#define N_ERPU                (1UL)  /**< Number of efuse read  protection Max  4 */
#define N_EWPU                (1UL)  /**< Number of efuse write protection Max  4 */
```

2. Configure each SWPU according to the number of SWPUs.

```
/***************************************
*   Application Protection
***************************************/

CY_SECTION(".cy_SFlash_app_prot") __USED static const cy_stc_si_app_prot_t cy_si_appprot =
{
    .objSize                = OBJECT_SIZE,             /* Application Protection Object Size (in bytes) */
    .n_fwpu                 = N_FWPU,                  /* Number of FWPU Max 16 */
    .fwpu0_adr.addr30       = 0x10000000,              /* Add region if you need */
    .fwpu0_size.region_size = 0x200,                   /* in bytes (multiple of 4) */
    .fwpu0_size.enable      = APP_PROT_ENABLE,         /* FWPU0 enable */
    .fwpu0_sl_att.urw       = APP_PROT_ALLOW,          /* FWPU0 Slave Attribute */
    .fwpu0_sl_att.prw       = APP_PROT_ALLOW,          /* FWPU0 Slave Attribute */
    .fwpu0_sl_att.ns        = APP_PROT_ALLOW,          /* FWPU0 Slave Attribute */
    .fwpu0_sl_att.pc_mask   = 0x00FF,                  /* FWPU0 Slave Attribute */
    .fwpu0_ms_att.urw       = APP_PROT_ALLOW,          /* FWPU0 Master Attribute */
    .fwpu0_ms_att.prw       = APP_PROT_ALLOW,          /* FWPU0 Master Attribute */
```

```
    .fwpu0_ms_att.ns        = APP_PROT_ALLOW,          /* FWPU0 Master Attribute */
    .fwpu0_ms_att.pc_mask   = 0x00FF,                  /* FWPU0 Master Attribute */
    .n_erpu                 = N_ERPU,                  /* Number of ERPU Max 4 */
    .erpu0_offset.offset    = 0x68,                      /* ERPU0 offset (Default) */
    .erpu0_size.region_size = 0x18,                      /* ERPU0 region size (Default) */
    .erpu0_size.enable      = APP_PROT_ENABLE,           /* ERPU0 enable (Default) */
    .erpu0_sl_att.urw       = APP_PROT_ALLOW,            /* ERPU0 Slave Attribute (Default) */
    .erpu0_sl_att.prw       = APP_PROT_ALLOW,            /* ERPU0 Slave Attribute (Default) */
    .erpu0_sl_att.ns        = APP_PROT_ALLOW,            /* ERPU0 Slave Attribute (Default) */
    .erpu0_sl_att.pc_mask   = 0x00FF,                    /* ERPU0 Slave Attribute (Default) */
    .erpu0_ms_att.urw       = APP_PROT_ALLOW,            /* ERPU0 Master Attribute (Default) */
    .erpu0_ms_att.prw       = APP_PROT_ALLOW,            /* ERPU0 Master Attribute (Default) */
    .erpu0_ms_att.ns        = APP_PROT_ALLOW,            /* ERPU0 Master Attribute (Default) */
    .erpu0_ms_att.pc_mask   = 0x00FF,                    /* ERPU0 Master Attribute (Default) */
    .n_ewpu                 = N_EWPU,                    /* Number of EWPU Max 4 */
    .ewpu0_offset.offset    = 0x68,                      /* EWPU0 offset (Default) */
    .ewpu0_size.region_size = 0x18,                      /* EWPU0 region size (Default) */
    .ewpu0_size.enable      = APP_PROT_ENABLE,           /* EWPU0 enable (Default) */
    .ewpu0_sl_att.urw       = APP_PROT_ALLOW,            /* EWPU0 Slave Attribute (Default) */
    .ewpu0_sl_att.prw       = APP_PROT_ALLOW,            /* EWPU0 Slave Attribute (Default) */
    .ewpu0_sl_att.ns        = APP_PROT_ALLOW,            /* EWPU0 Slave Attribute (Default) */
    .ewpu0_sl_att.pc_mask   = 0x00FF,                    /* EWPU0 Slave Attribute (Default) */
    .ewpu0_ms_att.urw       = APP_PROT_ALLOW,            /* EWPU0 Master Attribute (Default) */
    .ewpu0_ms_att.prw       = APP_PROT_ALLOW,            /* EWPU0 Master Attribute (Default) */
    .ewpu0_ms_att.ns        = APP_PROT_ALLOW,            /* EWPU0 Master Attribute (Default) */
    .ewpu0_ms_att.pc_mask   = 0x00FF,                    /* EWPU0 Master Attribute (Default) */
};
```

Table 6 shows details for each element.

Table 6. Elements Descriptions

Element	Description
.objeSize	Number of configured elements
.n_fwpu	Number of FWPUs
.n_erpu	Number of ERPUs
.n_ewpu	Number of EWPUs
.fwpu0_adr.addr30	Base address of FWPU0. 4-byte aligned.
.fwpu0_size.region_size	FWPU0 region size
.fwpu0_size.enable	FWPU0 enable. 0: Disable 1: Enable
.fwpu0/erpu0/ewpu0_sl_att.urw	FWPU0/EWPU0 slave attribute for user write restriction. 0: Prohibit 1: Allow EWPU0 slave attribute for user read restriction. 0: Prohibit 1: Allow
.fwpu0/erpu0/ewpu0_sl_att.prw	FWPU0/EWPU0 slave attribute for privileged write restriction. 0: Prohibit 1: Allow EWPU0 slave attribute for privileged read restriction. 0: Prohibit 1: Allow
.fwpu0/erpu0/ewpu0_sl_att.ns	FWPU0/EWPU0 slave attribute for non-secure write restriction. 0: Prohibit 1: Allow EWPU0 slave attribute for non-secure read restriction. 0: Prohibit 1: Allow

Element	Description
.fwpu0/erpu0/ewpu0_sl_att.pc_mask	FWPU0/ERPU0/EWPU0 slave attribute for PC restriction. The PC number corresponds to the bit number. 0: Prohibit 1: Allow (When pc_mask is 0x0005, PC0 and PC2 are allowed)
.fwpu0_ms_att.urw	FWPU0/EWPU0 master attribute for user write restriction. 0: Prohibit 1: Allow EWPU0 master attribute for user read restriction. 0: Prohibit 1: Allow
.fwpu0_ms_att.prw	FWPU0/EWPU0 master attribute for privileged write restriction. 0: Prohibit 1: Allow EWPU0 master attribute for privileged read restriction. 0: Prohibit 1: Allow
.fwpu0_ms_att.ns	FWPU0/EWPU0 master attribute for non-secure write restriction. 0: Prohibit 1: Allow EWPU0 master attribute for non-secure read restriction. 0: Prohibit 1: Allow
.fwpu0_ms_att.pc_mask	FWPU0/ERPU0/EWPU0 master attribute for PC restriction. The PC number corresponds to the bit number. 0: Prohibit 1: Allow (When pc_mask is 0x0005, PC0 and PC2 are allowed)
.erpu0/ewpu0_offset.offset	ERPU0 base address offset
.erpu0/ewpu0_size.region_size	ERPU0 region size
.erpu0/ewpu0_size.enable	ERPU0 enable. 0: Disable 1: Enable

Appendix G Normal Access Restriction

This project provides a configuration of the Normal Access Restriction (NAR). NAR determines DAP restrictions for NORMAL_PROVISIONED and SECURE_W_DEBUG. NAR is deployed in flash booting. NAR is stored in SFlash; they can be updated unlike the access restrictions in SECURE lifecycle stage. NAR has two elements: Normal Access Restrictions and Normal Dead Access Restrictions. See the “BootROM” chapter in Architecture TRM for NAR.

This project can configure the Normal Access Restrictions and Normal Dead Access Restrictions for M0+ DAP, M4 DAP, and System DAP.

G.1 Configuration

The following code shows the steps to configure normal access restriction. This section is located at 0x17001A00 in SFlash.

```
#define CY_SI_CM0_ENABLE                (0UL)  /**< CM0 ACCESS PORT ENABLE */
#define CY_SI_CM0_DISABLE_TMP           (1UL)  /**< CM0 ACCESS PORT TEMPORARY DISABLE */
#define CY_SI_CM0_DISABLE               (2UL)  /**< CM0 ACCESS PORT PERMANENTLY_DISABLE */
#define CY_SI_CM4_ENABLE                (0UL)  /**< CM4 ACCESS PORT ENABLE */
#define CY_SI_CM4_DISABLE_TMP           (1UL)  /**< CM4 ACCESS PORT TEMPORARY DISABLE */
#define CY_SI_CM4_DISABLE               (2UL)  /**< CM4 ACCESS PORT PERMANENTLY_DISABLE */
#define CY_SI_SYS_ENABLE                (0UL)  /**< SYS ACCESS PORT ENABLE */
#define CY_SI_SYS_DISABLE_TMP           (1UL)  /**< SYS ACCESS PORT TEMPORARY DISABLE */
#define CY_SI_SYS_DISABLE               (2UL)  /**< SYS ACCESS PORT PERMANENTLY_DISABLE */


/* Access Restriction */
#define CY_SI_NAR_NORMALACCESSRESTRICTION ((CY_SI_CM0_ENABLE << CY_SI_CM0_AP_POS) \
                                | (CY_SI_CM4_ENABLE << CY_SI_CM4_AP_POS) \
                                | (CY_SI_SYS_ENABLE << CY_SI_SYS_AP_POS)) \
                                | 0x80                                          /* Fixed value */

#define CY_SI_NAR_NORMALDEADACCESSRESTRICTION ((CY_SI_CM0_ENABLE << CY_SI_CM0_AP_POS) \
                                | (CY_SI_CM4_ENABLE << CY_SI_CM4_AP_POS) \
                                | (CY_SI_SYS_ENABLE << CY_SI_SYS_AP_POS))

CY_SECTION(".cy_SFlash_nar") __USED static const cy_stc_si_nar_t cy_nar =
{
    .nar         = CY_SI_NAR_NORMALACCESSRESTRICTION,       /* Normal Access Restrictions */
    .ndar        = CY_SI_NAR_NORMALDEADACCESSRESTRICTION,   /* Normal Dead Access Restrictions */
};
```

Table 7 shows details for each element.

Table 7. NAR setting

Element	Description
CY_SI_CM0/CM4/SYS_ENABLE	Corresponding DAP is enabled
CY_SI_CM0/CM4/SYS_DISABLE_TMP	Corresponding DAP is temporarily disabled. DAP can be re-enabled by application software.
CY_SI_CM0/CM4/SYS_DISABLE	Corresponding DAP is permanently disabled.

Note: NAR cannot be set less restrictive. For example, Disable setting cannot be changed to Enable setting.

Appendix H Sample APIs

The sample project **TVIIBE1M_RMA_si** has the following sample APIs and code.

- TransitionToSecure
- ReadUniqueID
- TransitionToRMA
- OpenRMA

H.1 TransitionToSecure API

This API transitions the lifecycle stage of the MCU to “SECURE” or "SECURE_W_DEBUG". This API validates the FACTORY_HASH, and programs the SECURE_HASH, secure access restrictions, and dead access restrictions into eFuse.

Execution of this API requires that the MCU lifecycle stage is in “NORMAL_PROVISIONED”. In addition, note that an MCU cannot return to “NORMAL_PROVISIONED” lifecycle stage after the transition of lifecycle stage to “SECURE” or “SECURE_W_DEBUG”.

This API has the following parameters.

- Debug: 1: Transition to SECURE_W_DEBUG lifecycle stage
  Other: Transition to SECURE lifecycle stage
- SECURE_ACCESS_RESTRICT: This parameter specifies the access restriction in Secure protection state
- DEAD_ACCESS_RESTRICT: This parameter specifies the access restriction in DEAD protection state

Parameter setting for Transition to SECURE

```
const cy_transitiontosecure_config_t TransitionToSecureConfig =
{
    .Debug = SECURE,
    .Secure_ACC_Rest  = SECURE_ACC_RESTRICT,
    .Dead_ACC_Rest = DEAD_ACC_RESTRICT,
};
```

Parameter setting for Transition to SECURE_W_DEBUG

```
const cy_transitiontosecurewithdebug_config_t TransitionToSecureWithDebugConfig =
{
    .Debug = SECURE_WITH_DEBUG,
    .Secure_ACC_Rest  = SECURE_ACC_RESTRICT,
    .Dead_ACC_Rest = DEAD_ACC_RESTRICT,
};
```

H.2 ReadUniqueID API

This API reads the unique ID of the die from SFlash. The unique ID is used for RMA lifecycle stage transition. Unique ID is returned to SRAM_SCRACH after running this API.

Table 8. Unique ID in SRAMSCRACH

SRAM_SCRATCH Address	Bits [31:28]	Bits [23:16]	Bits [15:8]	Bits [7:0]
0x00	Status Code 0xA0: Success 0xF0: Error	Unique ID_0		
0x04	Unique ID_1			
0x08	Unique ID_2			

H.3 TransitionToRMA API

This API transitions the lifecycle stage of the MCU to “RMA”. The TransitionToRMA API converts parts from SECURE, or SECURE_W_DEBUG to the RMA lifecycle stage. RMA lifecycle stage parts can be opened for analysis by running the "OpenRMA" API. The OpenRMA is used with Cypress; the customer must provide to Cypress a certificate for running the OpenRMA API. Running of this API requires the certificate and digital signature. In addition, the valid public key must be stored in SFlash.

```
Parameter setting for Transition to RMA.
// Transition To RMA and Open RMA parameters
#define OBJECT_SIZE             (0x00000014ul)
#define COMMND_ID_TRAN_TO_RMA   (0x120028F0ul)
#define COMMND_ID_OPEN_RMA      (0x120029F0ul)
#define UNIQUE_ID_0             (0x030dfe10ul) // Need to change per chip
#define UNIQUE_ID_1             (0xb1000a0aul) // Need to change per chip
#define UNIQUE_ID_2             (0x00130902ul) // Need to change per chip


uint32_t Digital_Signature_TransitionToRMA[64]
 =  {0x788a21b0ul,0x0f71d29cul,0x20a4f3c2ul,0x57852970ul,0xf46cdda9ul,0xc9cc67cdul,0x86f131f8ul,0xf5e65d33ul,
     0x1c5c37eful,0x9255fde0ul,0xe00503f2ul,0x5caf8b8cul,0xf0a00f81ul,0x67892d77ul,0x48097c34ul,0x43bf32ccul,
     0x69c94f85ul,0x6f04b237ul,0xf03eff12ul,0xea0fe3eaul,0x0af4b6eful,0x0465c648ul,0xfadc99b0ul,0x1f96ba9ful,
     0x5a41c3a5ul,0x9150a630ul,0x949d92b9ul,0x2fb33561ul,0x7ac2985dul,0x1bb1c3e9ul,0x808566b5ul,0xbac446e9ul,
     0x44ca41e3ul,0x08d2d73cul,0xfe8e745dul,0xf3e8c470ul,0xb39d66ecul,0x1e532e58ul,0x1f36814cul,0x22ee474cul,
     0x7d3175c0ul,0x4782bb62ul,0x5f41ad97ul,0x5282d700ul,0x2032d0e2ul,0x99a339e7ul,0x1e073d9dul,0xeccc2f44ul,
     0x12d2b206ul,0x1fd6da4aul,0x2d5805bdul,0x2f45140cul,0x077cad56ul,0x6005452cul,0xfd2e0a7ful,0x6d78a91dul,
     0xfca46879ul,0x6225e2a6ul,0x317ba835ul,0xc02d7263ul,0x8fcc5189ul,0xa5a57fb8ul,0x73a073d8ul,0x1126a6f7ul,};


uint32_t Digital_Signature_OpenRMA[64]
 =  {0x780a64a1ul,0x8cd185b5ul,0xdc4a1a7eul,0xbcf7469eul,0x576e0966ul,0x78afed5aul,0x0757346ful,0x873be620ul,
     0x4782f641ul,0x8ba52aa7ul,0x13c28b8ful,0xb74c22dful,0xdb21fd9ful,0xa43ce7f9ul,0x5f92c14cul,0xc1b10507ul,
     0xc66cd4d1ul,0x841408eeul,0xb039fe78ul,0xd9a11697ul,0x239cdc70ul,0xffca2921ul,0x8f85fca5ul,0x33206521ul,
     0xda894ed1ul,0xe62226acul,0x7a12b45bul,0xcb976045ul,0x1e1f821eul,0xb1070908ul,0x0468f9a8ul,0x626da06cul,
     0xf76dbd7bul,0xe2649263ul,0x042febbeul,0x6840d4c6ul,0x4c43514eul,0x96692416ul,0x8efaf699ul,0xccd7d830ul,
     0xf12c41c3ul,0x40f6da43ul,0xb8c0a11bul,0xc889a07bul,0x0c276c23ul,0x12bc0d44ul,0xb5604fdcul,0xf3baa0d5ul,
     0xf1bc02e4ul,0xc980dc32ul,0x1bf0fde7ul,0x82d997e4ul,0x8afbc242ul,0x0acff3cbul,0x682327bdul,0x8109e0c4ul,
     0x985cb1acul,0x717d4981ul,0x8c9eb0b8ul,0x54fee384ul,0x33ab3a3ful,0xfbcfc8fcul,0xe800bd77ul,0x7fdd6f7bul,};


const cy_transitiontorma_config_t TransitionToRMAConfig =
{
    .ObjectSize = OBJECT_SIZE,
    .CommandID  = COMMND_ID_TRAN_TO_RMA,
    .UniqueID_0 = UNIQUE_ID_0,
    .UniqueID_1 = UNIQUE_ID_1,
    .UniqueID_2 = UNIQUE_ID_2,
    .dataAddr = Digital_Signature_TransitionToRMA,
};

const cy_openrma_config_t OpenRMAConfig =
{
    .ObjectSize = OBJECT_SIZE,
    .CommandID  = COMMND_ID_OPEN_RMA,
    .UniqueID_0 = UNIQUE_ID_0,
    .UniqueID_1 = UNIQUE_ID_1,
    .UniqueID_2 = UNIQUE_ID_2,
    .dataAddr = Digital_Signature_OpenRMA,
};
```

H.4 Running the API

In this sample project, running the API can be selected by “ExecuteAPI” in the *main_cm4.c* source file. Table 9 lists the ExecuteAPI setting and running the API.

Table 9. ExecuteAPI Parameter

ExecuteAPI	Running API
= 1	TransitionToRMA
= 2	OpenRMA
= 3	TransitionToSecure (for SECURE)
= 4	TransitionToSecure (for SECURE_W_DEBUG)
Others	ReadUniqueID (Default: 0)

```
switch (ExecuteAPI) {
 case 1:
   status = Cy_Sys_TransitionToRMA(&sromContext, &TransitionToRMAConfig, CY_FLASH_DRIVER_NON_BLOCKING);
   break;
 case 2:
   status = Cy_Sys_OpenRMA(&sromContext, &OpenRMAConfig, CY_FLASH_DRIVER_NON_BLOCKING);
   break;
 case 3:
   status = Cy_Sys_TransitionToSecure(&sromContext, &TransitionToSecureConfig, CY_FLASH_DRIVER_NON_BLOCKING);
   break;
 case 4:
   status = Cy_Sys_TransitionToSecureWithDebug(&sromContext, &TransitionToSecureWithDebugConfig,
            CY_FLASH_DRIVER_NON_BLOCKING);
   break;
 default:
   status = Cy_Sys_ReadUniqueID(&sromContext, CY_FLASH_DRIVER_NON_BLOCKING);
   break;
}
```

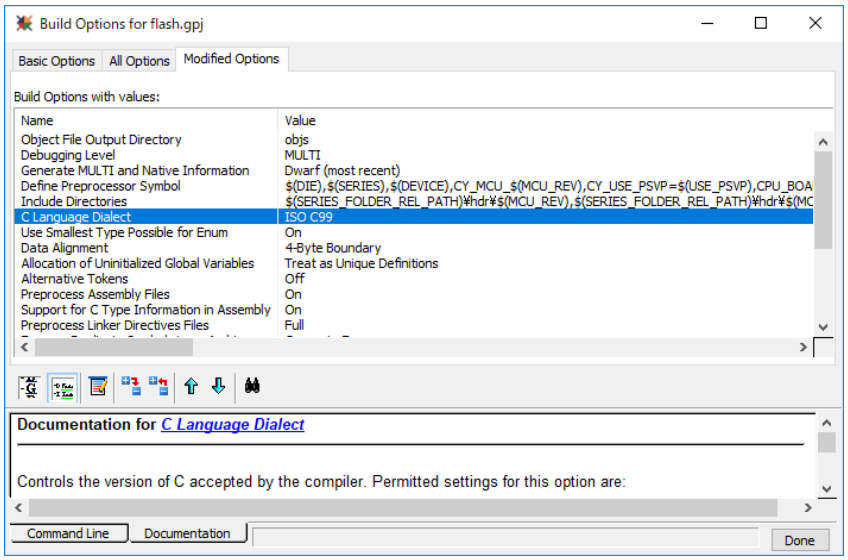
Appendix I Compile and Linker Options

The following options are required when building a secure boot using Green Hills MULTI.

- Compile Option: Set the C Language Dialect to ISO C99.

This setting is required to configure the secure image using the sample project.

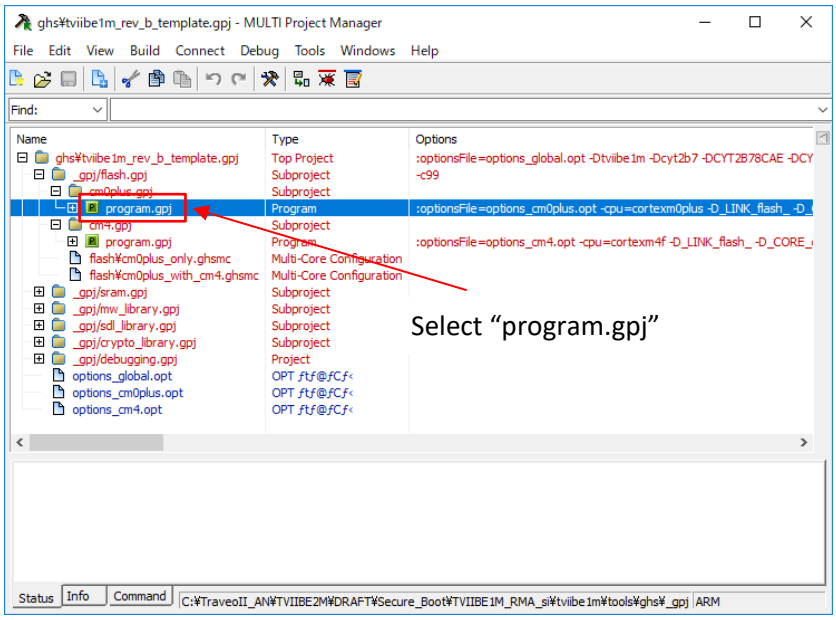
1. Select the target project.
2. Select **Edit** > **Set Build Options...**.



- Linker Option: Set the physical address offset from the Virtual Address to '0'.

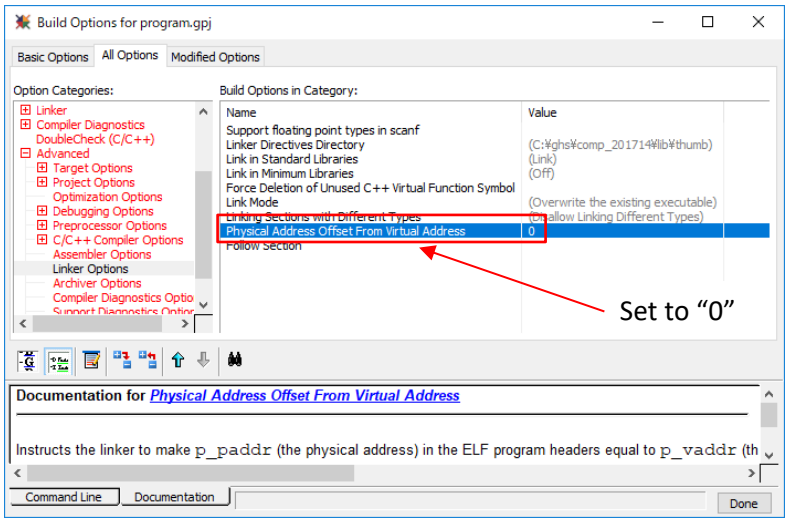
This setting is required to generate a digital signature using the *cymcuelftool.exe* tool.

1. Select **program.gpj** in **cm0plus.gpj**.



2. Select **Edit** > **Set Build Options…**, and then select the **All Options** tab.

3. In the **Option Categories** list, expand the **Advanced** group, and then select **Linker Options**.



4. Set the same configuration to **program.gpj** in *cm4.gpj*.

- Generate the SREC file.

As mentioned earlier, the blank area in the code must be ‘0’ for digital signature authentication. The following shows how to fill the blank area of the SREC file with ‘0’ for GHS MULTI. The SREC file is generated using the gsrec utility.

```
> gsrec.exe input_file (*.elf) -o out_file (*.srec) -bytes 16 -fill1 0x10000000 0x1000FEFF 0x00
```

Related Documents

The following are the Traveo II family series datasheets and Technical Reference Manuals. Contact Technical Support to obtain these documents.

- Device datasheet
  - CYT2B7 Datasheet 32-Bit Arm® Cortex®-M4F Microcontroller Traveo™ II Family
  - CYT2B9 Datasheet 32-Bit Arm® Cortex®-M4F Microcontroller Traveo™ II Family
  - CYT4BF Datasheet 32-Bit Arm® Cortex®-M7 Microcontroller Traveo™ II Family
  - CYT4DN Datasheet 32-Bit Arm® Cortex®-M7 Microcontroller Traveo™ II Family
- CYT2B Series
  - Traveo™ II Automotive Body Controller Entry Family Architecture Technical Reference Manual (TRM)
  - Traveo™ II Automotive Body Controller Entry Registers Technical Reference Manual (TRM) for CYT2B7
  - Traveo™ II Automotive Body Controller Entry Registers Technical Reference Manual (TRM) for CYT2B9
- CYT4B Series
  - Traveo™ II Automotive Body Controller High Family Architecture Technical Reference Manual (TRM)
  - Traveo™ II Automotive Body Controller High Registers Technical Reference Manual (TRM)
- CYT4D Series
  - Traveo™ II Automotive Cluster 2D Family Architecture Technical Reference Manual (TRM)
  - Traveo™ II Automotive Cluster 2D Registers Technical Reference Manual (TRM)

Other References

Contact Technical Support to obtain the code snippets and tools used in this application note. Note that the obtained sample program does not conform to automotive standards, so it is used only as an example for secure system configuration and cannot be used for production purposes.

Document History

Document Title: AN228680 - Secure System Configuration in Traveo™ II Family

Document Number: 002-28680

Revision	ECN	Submission Date	Description of Change
**	6749608	03/24/2020	New Application Note.

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 | Code Examples | Projects | Videos | Blogs | Training | Components

Technical Support

cypress.com/support

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