Application Note

Document No.: AN1139

G32R501 Secure Boot Application Note

Version: V1.0

1 Introduction

This application note aims to introduce the content about secure boot of G32R501 chip, including the design principles of secure boot application programs and how to use the secure boot routines provided by G32R5xx SDK.

Contents

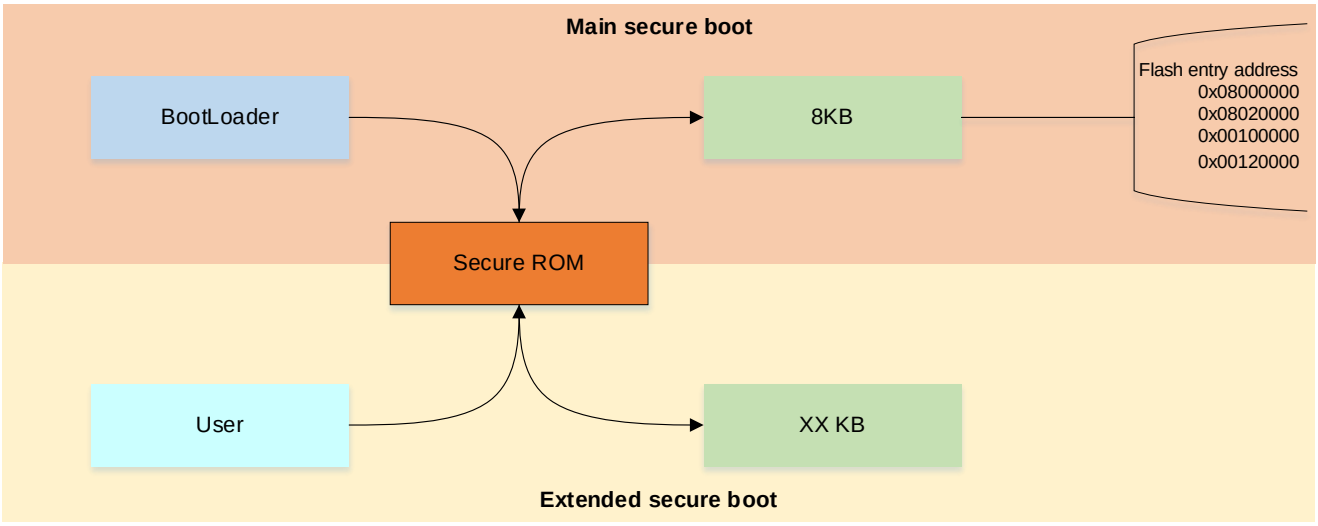
2 Overview of G32R5xx Secure Boot

The secure boot function of G32R5xx is implemented by the cured BootROM code of the chip, and a 128-bit AES-CMAC authentication algorithm is used to authenticate the first 8KB data of user firmware stored in Flash. Only after the authentication is passed, will it jump to execute the user application.

The authentication initiated by BootROM for the first 8KB of user firmware is called “primary secure boot”. Accordingly, the authentication beyond the 8KB range is called "extended secure boot", and this process is selectively initiated by user application code.

The Flash sector where the primary secure boot is checked must be set to Zone1-ExeOnly permission. The Flash sector where the extended secure boot is checked can be configured with three permissions of Unsecure, Zone1-Secure, and Zone1-ExeOnly. That is to say, the encryption permission of the Flash sector where the CMAC authentication application is located is not allowed to be configured as Zone2 permission.

Figure 1 Primary Secure Boot and Extended Secure Boot Check Diagram



3 Secure Boot Option

The secure boot function of G32R5xx only supports boot from CPU0. Table 1 lists all entry addresses for secure boot and the corresponding BootMode values, as well as the locations where Golden CMAC Tag must be stored under different entry addresses.

Table 1 Secure Boot Option Configuration Table

Option	BootMode value	Flash entry point	Flash block/sector	Golden CMAC Tag storage location
0	0x0A	0x08000000	Bank 0, Sector 0 (Busmatrix IF)	0x08000008
1	0x2A	0x00100000	Bank 0, Sector 0 (ITCM IF)	0x00100008
2	0x4A	0x08020000	Bank 1, Sector 0(Busmatrix IF,DualBank Mode	0x08020008
3	0x6A	0x00120000	Bank 1, Sector 0(ITCM IF,BualBank Mode)	0x00120008

Note: The corresponding entry sectors of secure boot must be configured with Zone1-ExeOnly permission.

4 How to Design Secure Boot Applications

The applications that require secure boot must meet the requirements described in this chapter to ensure proper check of the application and ensure its proper running.

4.1 Golden CMAC Tag Storage Location

The Golden CMAC Tag value is the expected Tag value stored by the user in the firmware for CMAC authentication, with a size of 16 bytes. For the primary secure boot, the start address for the storage of the Golden CMAC Tag must be located at 8 bytes after the entry address corresponding to the secure boot. For example, if the entry address for secure boot check is 0x08000000, the start address for the storage of Golden CMAC Tag shall be 0x08000008.

For the extended secure boot of which users want to customize the Flash sector interval of the check, there are no restrictions on the start address for storing the Golden CMAC Tag value, but the following requirements must be met:

（1） The Golden CMAC Tag value for the extended secure boot must not be within the check range of the primary secure boot.

（2） The start address of storage for the Golden CMAC Tag value of the extended secure boot must be 4 bytes-aligned.

（3） The storage area for Golden CMAC Tag value for extended secure boot must be within the range where CMAC validation is performed. For example, if the size of the entire Flash is checked, the Golden CMAC Tag must be stored within the Flash range.

4.2 Requirements for Encrypted Sector Settings of Applications to be Checked

When the Flash sector where the program that requires CMAC authentication is located is set with the encryption permission, in this area the program can only be executed by the CPU, and the CPU is not allowed to read the data of the Flash sector. Therefore, if the data that needs to be read by CPU is stored in Flash, the corresponding Flash sector may not be set with encryption permission.

Usually, the data that the CPU needs to read from Flash includes read-only data segments (e.g. global variables modified with const), and the initial values of global variables with non-zero initial values. The initial values of these variables are stored in the Flash area and copied to the RAM area when the MCU starts. Therefore, the code areas that are not allowed to be set with encryption permission include:

(1) CONST (RO-DATA): Read-only data area.

(2) RW: Readable and writable data segments. The global variables with non-zero initial values are stored in the code together with other data during compilation, and the initial values will be copied from Flash to RAM after the power supply is turned on.

It can be seen from this that the only segment that is allowed to be encrypted and stored in Flash is the text segment, namely the code segment. The Flash sectors where other data segments are located are not allowed to be encrypted.

4.3 Application Design Example, Taking MDK-ARM Environment as an Example

Based on the introduction in Sections 4.1 and 4.2, this section takes the MDK-ARM environment as an example to illustrate how to design applications that meet secure boot requirements.

4.3.1 Design example of Golden CMAC Tag value storage location

Section 4.1 above introduces the requirements for the storage location of Golden CMAC Tag values, and the following introduces how to implement these requirements in the code.

1. Design of Golden CMAC Tag value storage location for primary secure boot:

Due to the regulations of the ARM Cortex-M architecture, the first 8 bytes of the start address must store the values of the SP and PC registers. Therefore, the Golden CMAC Tag value is specified to be stored after the SP and PC register location, that is, the location with an offset by 8 bytes from the entry address.

Moreover, the sector where the primary secure boot entry address is located must be configured with Zone1-ExeOnly permission. To ensure that the interrupt vector table is not encrypted, other interrupt vectors cannot be stored in the entry sector.

Based on the above two reasons, the solutions of two interrupt vector tables are designed. The first interrupt vector table is used to store the values of SP and PC registers, and the Golden CMAC Tag values; the second interrupt vector table is used to store all other interrupt vectors.

Design an interrupt vector table to store the Golden CMAC Tag values, and the code example is

```
const PINT __VECTOR_TABLE_CMAC[6] __VECTOR_TABLE_ATTRIBUTE = {
    [0]   = (PINT)(&__INITIAL_SP),    // Initial Stack Pointer
    [1]   = Reset_Handler,            // Reset Handler (Code Start)
    [2]   = (PINT)0xFFFFFFFF,         // CMAC Tag 0
    [3]   = (PINT)0xFFFFFFFF,         // CMAC Tag 1
    [4]   = (PINT)0xFFFFFFFF,         // CMAC Tag 2
    [5]   = (PINT)0xFFFFFFFF,         // CMAC Tag 3

};
```

as follows:

This interrupt vector table must be stored at the start address.

The other interrupt vector table stores all interrupt vectors, and its storage location is not specified, but the sector where the vector table is located is not allowed to be set with

```
const PINT __VECTOR_TABLE[512] = {
    [0]   = (PINT)(&__INITIAL_SP),    // Initial Stack Pointer
    [1]   = Reset_Handler,            // Reset Handler (Code Start)
    [2]   = NMI_Handler,              // -14 NMI Handler
    [3]   = HardFault_Handler,        // -13 Hard Fault Handler
    [4]   = MemManage_Handler,        // -12 MPU Fault Handler
    [5]   = BusFault_Handler,         // -11 Bus Fault Handler
    [6]   = UsageFault_Handler,       // -10 Usage Fault Handler
    [7]   = SecureFault_Handler,      //  -9 Secure Fault Handler
    [11]  = SVC_Handler,              //  -5 SVCall Handler
    [12]  = DebugMon_Handler,         //  -4 Debug Monitor Handler
    [14]  = PendSV_Handler,           //  -2 PendSV Handler
    [15]  = SysTick_Handler,          //  -1 System Tick Handler

    //
    // Interrupts
    //
    [27]  = DCCOMP_Handler,           //  11 DCCOMP Handler
    [29]  = TIMER1_Handler,           //  13 CPU Timer 1 Handler
    [30]  = TIMER2_Handler,           //  14 CPU Timer 2 Handler
    [32]  = RAM_Handler,              //  16 RAM Handler
    [33]  = IPC_TE0_Handler,          //  17 IPC TE0 Handler
    [34]  = IPC_TE1_Handler,          //  18 IPC TE1 Handler
    [35]  = IPC_TE2_Handler,          //  19 IPC TE2 Handler
    [36]  = IPC_TE3_Handler,          //  20 IPC TE3 Handler
    [37]  = IPC_RF0_Handler,          //  21 IPC RF0 Handler
    [38]  = IPC_RF1_Handler,          //  22 IPC RF1 Handler

    ......
};
```

encryption permission. The code example is as follows (excerpt):

Not all interrupt vectors are displayed here. For detailed code, please refer to the corresponding example program of G32R5xx SDK.

2. Design of Golden CMAC Tag value storage location for extended secure boot:

Users can authenticate the data exceeding 8KB, and the authentication initiated by users belongs to extended secure boot. According to the requirements for storage position of the Golden CMAC Tag values of the extended secure boot mentioned above, the corresponding

code example is as follows:

```
const uint32_t cmac_all[] \
__attribute__((__used__, section(".ARM.__at_0x08008000"))) =
{
    0xFFFFFFFF,
    0xFFFFFFFF,
    0xFFFFFFFF,
    0xFFFFFFFF,

};
```

This array is designed to ensure that a specific position is occupied in the bin file generated by compilation, to prevent the compiler from using the space used for storing the Golden CMAC Tag values to store other data. The address of 0x08008000 is the start storage address of Golden CMAC Tag.

4.3.2 Link file design example

4.2Section introduction: For the compiled bin, except for the code segment that can be set with the encryption permission, other segments are not allowed to be set with encryption permission. Therefore, the link files of secure boot applications must be redesigned to store the areas not allowed to be encrypted in fixed Flash sector.

Note: Do not set the Flash sector where the segment area not allowed to be encrypted is located with encryption permission.

In the example of the sct file in this section, the LR_SECURE_ROM area only stores the code segment and the first interrupt vector table array, and the Flash sector where this area is located can be set with encryption permission. The area where LR_UNSECURE_ROM is located contains read-only data segments, __main function, RW data segment, and other contents, and the Flash sector where this area is located is not allowed to be encrypted. If this area is encrypted, the program will be caused to fail to execute properly.

The following is an example of the sct link file of MDK-ARM, and the link file design example is

```
// This area allows users to be set with encryption permissions.
LR_SECURE_ROM __RO_BASE __RO_SIZE {
  ER_SECURE_ROM __RO_BASE __RO_SIZE {
   *.o (RESET, +First)
   .ANY (+RO)
   .ANY (+XO)
  }
}

// This area does not allow users to be set with encryption
permissions.
LR_UNSECURE_ROM __CPU0_UNSECURE_ROM_BASE __CPU0_UNSECURE_ROM_SIZE {
  ER_UNSECURE_ROM __CPU0_UNSECURE_ROM_BASE __CPU0_UNSECURE_ROM_SIZE {
    *(InRoot$$Sections)
    .ANY (+CONST)
  }

  RW_RAM_CPU0_ITCM __CPU0_ITCM_BASE __CPU0_ITCM_SIZE {
    .ANY (itcm.instruction)
    .ANY (itcm.ramfunc)
    .ANY (+RW +ZI)
  }

#if __HEAP_SIZE > 0
  ARM_LIB_HEAP __HEAP_BASE EMPTY __HEAP_SIZE {   ; Reserve empty
region for heap
  }
#endif

  ARM_LIB_STACK __STACK_TOP EMPTY - __STACK_SIZE {   ; Reserve empty
region for stack
  }

}
```

as follows (excerpt):

For all the content of the file, please refer to the sct file of corresponding secure boot routine of G32R5xx SDK.

5 How to Use G32R5xx Secure Boot Function

5.1 Operation Process of Secure Boot

To implement secure boot, in addition to following the design principles introduced in Chapter 4, the following operational process shall also be followed when designing the application.

(1) Configure the starting sector permission of secure boot as Zone1 ExeOnly permission.

(2) Configure the CMACKEY value of the chip. The CMACKEY value of the chip is stored in the OTP area, with an address of 0x0810B020 and a length of 128 bytes. When the chip leaves the factory, the CMACKEY value is considered to be full FF by default.

(3) For the original bin file of the application compiled and generated by users, according to the CMACKEY value set by users, use the Geehy_Bin tool to generate a bin file with the Golden CMAC Tag value.

(4) Burn the bin file with the Golden CMAC Tag value and download it to the chip.

(5) According to the secure boot option configuration table in Table 1, set the required BootMode value (in debugging phase, it is recommended to use the simulated BootMode value to set it to the secure boot mode).

(6) Press the Reset button or power on for reset to execute the application.

5.2 Introduction to Use of Geehy_Bin Tools

After the user compiles the application and generates the original bin file, the Geehy_Bin tool can be used to generate the bin file with the Golden CMAC Tag value for secure boot.

This section only introduces how to use the Geehy_Bin tool to generate the bin file with the Golden CMAC Tag value. For more information on the use of this tool, please refer to the *AN1131_G32R501 Tool User Manual*.

5.2.1 key.txt and cmd files

To use the Geehy_Bin tool to generate the bin file with the Golden CMAC Tag value, the user needs to provide a text file with the CMACKEY value, and the check length information of the primary secure boot or extended secure boot. In the G32R5xx SDK, examples of these files have already been provided under the utilities/geehy_tool directory, including three files, i.e. key.txt, test_major.cmd, and test_extend.cmd.

- key.txt file.

This file is used to specify the CMACKEY value set by the user. The example content is as

```
0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
```

follows, which is the default CMACKEY value of the chip.

Note: If the CMACKEY value of the chip is modified, the content of this file must correspond to the CMACKEY value set by the user inside the chip.

- test_major.cmd file

  The test_major.cmd file is used to specify the entry address and check length of the primary secure boot, but the check length of the primary secure boot is fixed at 0x2000. The

```
/* CPU1 Flash sectors */

/* HEX directive required by HEX utility to generate the golden
CMAC tag */
/* with one entry that represents all the allocated flash memory */
ROMS
{
  FLASH: o=0x08000000 l=0x2000, fill = 0xFFFF /* If fill not
specified, then default is all 0s */
}
```

  example content of the test_major.cmd file is as follows:

  Where, o=0x08000000 is used to specify the entry address of the primary secure boot check, and this value must be consistent with the entry address corresponding to the BootMode value set by the user. l=0x2000 is used to specify the check length of the primary secure boot. It is important to that the check length of the primary secure boot is fixed at 0x2000.

- test_ extend.cmd file

  The test_extend.cmd file is used to specify the start address and check length of user-

```
/* CPU1 Flash sectors */

/* HEX directive required by HEX utility to generate the golden
CMAC tag */
/* with one entry that represents all the allocated flash memory */
ROMS
{
  FLASH: o=0x08000000 l=0x10000, fill = 0xFFFF /* If fill not
specified, then default is all 0s */
}
```

  defined check. The content of the file example is as follows:

Wherein, o=0x08000000 is used to specify the start address of the extended secure boot (i.e. user-defined check), and l=0x10000 is used to specify the check length of the extended secure boot. The check start address and length specified for the extended secure boot must correspond to the check start address and length set by the application. In addition, the check range for the extended secure boot can include the check range of 0x2000 for the primary secure boot.

5.2.2 Command Use of Geehy_Bin tool

- Generate the bin file command of the primary secure boot

  Geehy_Bin.exe test_major.cmd -m --load_image --cmac=key.txt project.bin -tag=0x08000008 -o project_cmac_major.bin

  Where, -tag=0x08000008 is used to specify the start address for storage of the Golden CMAC Tag value generated by the operation. For the primary secure boot, the start address for storage of the Golden CMAC Tag value is at the location with an offset by 8 bytes from the entry address.

- Generate the bin file command of the extended secure boot

  Geehy_Bin.exe test_extend.cmd -e --load_image --cmac=key.txt project_cmac_major.bin -tag=0x08008000 -load=0x08000000 -o project_cmac_extend.bin

  For the extended secure boot, the start address specified by the user using the -tag parameter to store the Golden CMAC Tag value must correspond to the address set by the application to store this value one on one. Moreover, its storage location is not allowed to be within the check range of the primary secure boot.

The extended secure boot is selectively initiated by the user. If both the primary secure boot and the extended secure boot exist, the user must first generate a bin file for the primary secure boot, and then use the bin file to generate a bin file in which both the primary secure boot and extended secure boot exist.

5.3 Use of G32R5xx SDK Secure Boot Routine

The G32R5xx SDK has provided a secure boot sample application. This secure boot application includes a check example for the primary secure boot of 8KB, and an example of how to check the code that exceeds the 8KB range.

This example program is located at:
driverlib\g32r501\examples\eval\boot\boot_ex3_cpu0_secure_flash.

This example assumes that the Flash sector for authentication has been correctly configured with encryption permission (the entry address sector must be set with Zone1 EXEONLY permission), and the default CMACKEY is used for authentication. For how to set Flash sector permission and modify CMACKEY, please refer to the relevant routines and introductions of the

DCS module.

The MDK-ARM environment is taken as an example below to introduce how to run this example program.

5.3.1 Generate the bin file with Golden CMAC Tag value

Geehy provides the tool for generating the bin file with the Golden CMAC Tag value. The following introduces how to use the Geehy tool to generate this bin file.

(1) Generate the bin file without the Golden CMAC Tag value from the hex file

Since the sct file of the secure boot application has multiple loading domains, it is impossible to use the bin file that comes with MDK-ARM to generate a completed bin file, but it generates a scattered bin file. So the hex2bin tool shall be used to generate the bin file, as shown in Figure 2:

Figure 2 Generate the Bin File without Golden CMAC Tag Value

```
D:\geehy_bin>hex2bin.exe project.hex
hex2bin v2.5, Copyright (C) 2017 Jacques Pelletier & contributors

Allocate_Memory_and_Rewind:
Lowest address:   08000000
Highest address:  080417FF
Starting address: 08000000
Max Length:       268288

Binary file start = 08000000
Records start     = 08000000
Highest address   = 080417FF
Pad Byte          = FF
```

(2) Use Geehy tool to generate the bin file with Golden CMAC Tag value

First, it is necessary to generate a bin file with the Golden CMAC Tag value for the primary secure boot. The command is as follows:

Geehy_Bin.exe test_major.cmd -m --load_image --cmac=key.txt project.bin -tag=0x08000008 -o project_cmac_major.bin

Figure 3 Generate the Bin File with Golden CMAC Tag Value

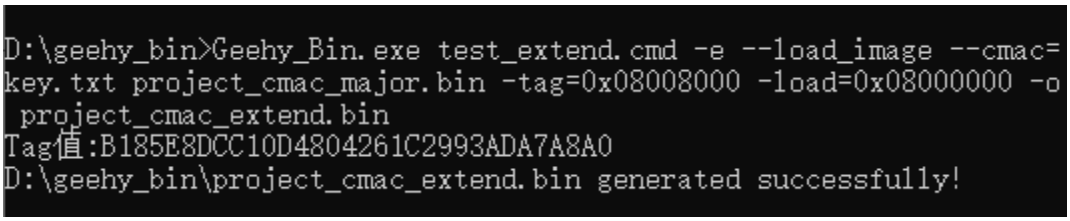
```
D:\geehy_bin>Geehy_Bin.exe test_major.cmd -m --load_image --cmac=key.txt
project.bin -tag=0x08000008 -o project_cmac_major.bin
Tag值:0B5A988D69B5281E98180B5D2C528883
D:\geehy_bin\project_cmac_major.bin generated successfully!
```

Finally, use the generated bin file with the Golden CMAC Tag value for the primary secure boot

to generate the bin file with the Golden CMAC Tag value for the extended secure boot. The command is as follows:

Geehy_Bin.exe test_extend.cmd -e --load_image --cmac=key.txt project_cmac_major.bin -tag=0x08008000 -load=0x08000000 -o project_cmac_extend.bin

Figure 4 Generate the Bin File with Golden CMAC Tag Value for Extended Secure Boot



First, generate the bin file with the Golden CMAC Tag value for the primary secure boot, and then use this bin file to generate the bin file with the Golden CMAC Tag value for the extended secure boot.

Note: The tools and corresponding files used in this section can be obtained from the G32R5xx SDK.

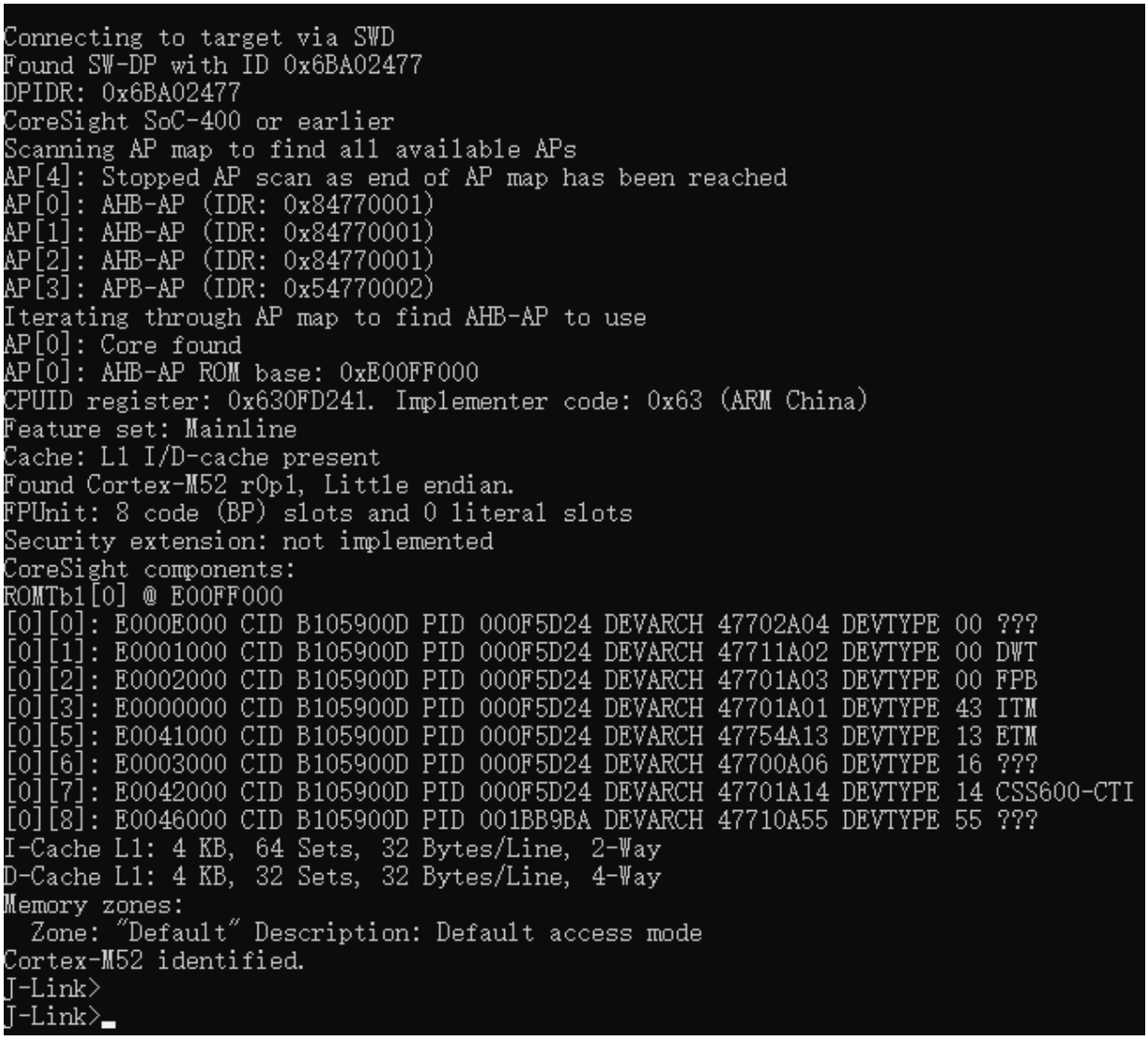
5.3.2 Change the boot mode to secure boot

The default boot mode of the chip is not secure boot, and users need to configure the boot mode to the secure boot mode themselves. The user can configure the Boot mode by modifying OTP and simulating the boot mode. The method of modifying OTP can only be modified once, and it is recommended to use the method of simulating the boot mode to modify the Boot mode in debugging phase. The following introduces how to use the method of simulating boot to change the boot mode to secure boot.

(1) First, use the Jlink debugger to connect to the development board.

(2) Use the Jlink Commander command line tool to connect the chip, and after successful connection, it will be as shown in Figure Figure 5.

Figure 5 Successfully Connection of Jlink Commander to G32R501 Window



Note: The Jlink Commander used must be the version that already supports the Cortex-M52 core. The version used in this demonstration is V7.96h. In addition, the Jlink debugger used also needs to support debugging of the Cortex-M52 core.

(3) Configure the boot mode to a secure boot mode with an entry address of 0x08000000, with a BootMode value of 0x0A. Enter the command on the Jlink Commander command line tool:

w4 0x50020000 0x5AFFFFFF

w4 0x50020004 0xFFFFFF0A

w4 0x50020008 0xFFFFFFFF

5.3.3 Download and run routines

The JFlash tool can be used to download the bin file generated earlier with the Golden CMAC

Tag value for the primary and extended secure boots.

Note: The G32R5xx chip provides encryption protection for DCS module, which needs to be decrypted before programming of Flash. On how to decrypt, please refer to the introduction to DCS module.

After the program is downloaded, press the reset button (please do not power off, or the previously configured boot mode will be lost), and observe the flashing of the LED lights on the development board. Judge whether the check is passed based on the flashing of the LED, as follows:

- If LED2 and LED3 are off, it indicates secure boot failed
- If LED2 or LED3 flashes, it indicates the secure boot is passed, and the user's extended CMAC check failed.
- If LED2 and LED3 flash, it indicates the secure boot is passed, and the user's extended CMAC check is passed.

6 Revision

Table 2 Document Revision History

Date	Version	Change History
January, 2025	1.0	New

Statement

This document is formulated and published by Geehy Semiconductor Co., Ltd. (hereinafter referred to as “Geehy”). The contents in this document are protected by laws and regulations of trademark, copyright and software copyright. Geehy reserves the right to make corrections and modifications to this document at any time. Read this document carefully before using Geehy products. Once you use the Geehy product, it means that you (hereinafter referred to as the “users”) have known and accepted all the contents of this document. Users shall use the Geehy product in accordance with relevant laws and regulations and the requirements of this document.

1. Ownership

This document can only be used in connection with the corresponding chip products or software products provided by Geehy. Without the prior permission of Geehy, no unit or individual may copy, transcribe, modify, edit or disseminate all or part of the contents of this document for any reason or in any form.

The “极海” or “Geehy” words or graphics with “®” or “™” in this document are trademarks of Geehy. Other product or service names displayed on Geehy products are the property of their respective owners.

2. No Intellectual Property License

Geehy owns all rights, ownership and intellectual property rights involved in this document.

Geehy shall not be deemed to grant the license or right of any intellectual property to users explicitly or implicitly due to the sale or distribution of Geehy products or this document.

If any third party’s products, services or intellectual property are involved in this document, it shall not be deemed that Geehy authorizes users to use the aforesaid third party’s products, services or intellectual property. Any information regarding the application of the product, Geehy hereby disclaims any and all warranties and liabilities of any kind, including without limitation warranties of non-infringement of intellectual property rights of any third party, unless otherwise agreed in sales order or sales contract.

3. Version Update

Users can obtain the latest document of the corresponding models when ordering Geehy products.

If the contents in this document are inconsistent with Geehy products, the agreement in the sales order or the sales contract shall prevail.

4. Information Reliability

The relevant data in this document are obtained from batch test by Geehy Laboratory or cooperative third-party testing organization. However, clerical errors in correction or errors caused by differences in testing environment may occur inevitably. Therefore, users should understand that Geehy does not bear any responsibility for such errors that may occur in this document. The relevant data in this document are only used to guide users as performance parameter reference and do not constitute Geehy's guarantee for any product performance.

Users shall select appropriate Geehy products according to their own needs, and effectively verify and test the applicability of Geehy products to confirm that Geehy products meet their own needs, corresponding standards, safety or other reliability requirements. If losses are caused to users due to user's failure to fully verify and test Geehy products, Geehy will not bear any responsibility.

5. Legality

USERS SHALL ABIDE BY ALL APPLICABLE LOCAL LAWS AND REGULATIONS WHEN USING THIS DOCUMENT AND THE MATCHING GEEHY PRODUCTS. USERS SHALL UNDERSTAND THAT THE PRODUCTS MAY BE RESTRICTED BY THE EXPORT, RE-EXPORT OR OTHER LAWS OF THE COUNTRIES OF THE PRODUCTS SUPPLIERS, GEEHY, GEEHY DISTRIBUTORS AND USERS. USERS (ON BEHALF OR ITSELF, SUBSIDIARIES AND AFFILIATED ENTERPRISES) SHALL AGREE AND PROMISE TO ABIDE BY ALL APPLICABLE LAWS AND REGULATIONS ON THE EXPORT AND RE-EXPORT OF GEEHY PRODUCTS AND/OR TECHNOLOGIES AND DIRECT PRODUCTS.

6. Disclaimer of Warranty

THIS DOCUMENT IS PROVIDED BY GEEHY "AS IS" AND THERE IS NO WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE, TO THE EXTENT PERMITTED BY APPLICABLE LAW.

GEEHY'S PRODUCTS ARE NOT DESIGNED, AUTHORIZED, OR WARRANTED FOR USE AS CRITICAL COMPONENTS IN MILITARY, LIFE-SUPPORT, POLLUTION CONTROL, OR HAZARDOUS SUBSTANCES MANAGEMENT SYSTEMS, NOR WHERE FAILURE COULD RESULT IN INJURY, DEATH, PROPERTY OR ENVIRONMENTAL DAMAGE.

IF THE PRODUCT IS NOT LABELED AS "AUTOMOTIVE GRADE," IT SHOULD NOT BE CONSIDERED SUITABLE FOR AUTOMOTIVE APPLICATIONS. GEEHY ASSUMES NO LIABILITY FOR THE USE BEYOND ITS SPECIFICATIONS OR GUIDELINES.

THE USER SHOULD ENSURE THAT THE APPLICATION OF THE PRODUCTS COMPLIES WITH ALL RELEVANT STANDARDS, INCLUDING BUT NOT LIMITED TO SAFETY, INFORMATION SECURITY, AND ENVIRONMENTAL REQUIREMENTS. THE USER ASSUMES FULL RESPONSIBILITY FOR THE SELECTION AND USE OF GEEHY PRODUCTS. GEEHY WILL BEAR NO RESPONSIBILITY FOR ANY DISPUTES ARISING FROM THE SUBSEQUENT DESIGN OR USE BY USERS.

7. Limitation of Liability

IN NO EVENT, UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL GEEHY OR ANY OTHER PARTY WHO PROVIDES THE DOCUMENT AND PRODUCTS "AS IS", BE LIABLE FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, DIRECT, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE DOCUMENT AND PRODUCTS (INCLUDING BUT NOT LIMITED TO LOSSES OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY USERS OR THIRD PARTIES). THIS COVERS POTENTIAL DAMAGES TO PERSONAL SAFETY, PROPERTY, OR THE ENVIRONMENT, FOR WHICH GEEHY WILL NOT BE RESPONSIBLE.

8. Scope of Application

The information in this document replaces the information provided in all previous versions of the document.